[Bootcamp: UCD-VIRT-DATA-PT-03-2022-U-B-MW](https://courses.bootcampspot.com/courses/1225)

[3.0.1: The Power of Python](https://courses.bootcampspot.com/courses/1225)

Immersive Reader

**3.0.1**

**The Power of Python**

[Bootcamp: UCD-VIRT-DATA-PT-03-2022-U-B-MW](https://courses.bootcampspot.com/courses/1225)

[3.0.2: Module 3 Roadmap](https://courses.bootcampspot.com/courses/1225)

Immersive Reader

**3.0.2**

# Module 3 Roadmap

## Looking Ahead

In this module, you will learn how to use the programming language Python to:

* Create scripts that read, write, and store data from files or in arrays.
* Perform mathematical operations.
* Use decision statements, logical operations for complex comparisons, and repetition statements to run code more than once.
* Produce screen outputs.

Throughout this module, you will need to apply your knowledge of variables, arrays, conditionals, and loops, as well as install and use new software.

## What You Will Learn

By the end of this module, you should be able to:

* Navigate to folders on your computer using the command line.
* Read and extract data from CSV files.
* Determine the difference between Python data types, like integers, floating-point decimal numbers, and strings.
* Perform mathematical operations using data types.
* Declare variables using different data types.
* Create and use data structures like lists, tuples, and dictionaries.
* Create and use decision and repetition statements.
* Create and use Boolean and logical operators.
* Write data to an output file and print the file.

## Planning Your Schedule

Here's a quick look at the lessons and assignments you'll cover in this module. You can use the time estimates to help pace your learning and plan your schedule.

* Introduction to Module 3 (15 minutes)
* Get Started with the Command Line and Python (1 hour)
* Python Practice (3 hours)
* Open the Election Results (30 minutes)
* Get Started with File Processing (3 hours)
* Analyze the Election Data (2 hours)
* Finalize the Election Results (1 hour)
* Application (5 hours)

[Previous](https://courses.bootcampspot.com/courses/1225/modules/items/497072)[Next](https://courses.bootcampspot.com/courses/1225/modules/items/497080)

© 2020 - 2022 Trilogy Education Services, a 2U, Inc. brand. All Rights Reserved.

[Bootcamp: UCD-VIRT-DATA-PT-03-2022-U-B-MW](https://courses.bootcampspot.com/courses/1225)

[3.0.3: Getting Ready for Virtual Class](https://courses.bootcampspot.com/courses/1225)

Immersive Reader

**3.0.3**

# Getting Ready for Virtual Class

Let’s make sure you’re prepared to join this week’s virtual classes! You’ll have the opportunity to practice your new skills with your instructional team, as well as ask any questions that you may have regarding this module’s content.

Please be sure to download the starter files below and have them ready on your computer prior to class. You do not need to complete any of the activities prior to class, but feel free to review the material ahead of time to see some of what will be covered.

## Virtual Class 1 (Required)

To make the most out of class, aim to do the following before you meet:

* Complete Lessons 1 and 2.
* Install Python.
* Download the following ZIP file, which contains all resources for this class:

[3-1-Student-Resources.zip (Links to an external site.)](https://2u-data-curriculum-team.s3.amazonaws.com/data-viz-online-lesson-plans/03-Lessons/3-1-Student-Resources.zip)

## Virtual Class 2 (Required)

To make the most out of class, aim to do the following before you meet:

* Complete Lessons 3, 4, 5 and 6.
* Review the Challenge instructions.
* Download the following ZIP file, which contains all resources for this class:

[3-2-Student-Resources.zip (Links to an external site.)](https://2u-data-curriculum-team.s3.amazonaws.com/data-viz-online-lesson-plans/03-Lessons/3-2-Student-Resources.zip)

[Previous](https://courses.bootcampspot.com/courses/1225/modules/items/497076)[Next](https://courses.bootcampspot.com/courses/1225/modules/items/497085)

© 2020 - 2022 Trilogy Education Services, a 2U, Inc. brand. All Rights Reserved.

[Bootcamp: UCD-VIRT-DATA-PT-03-2022-U-B-MW](https://courses.bootcampspot.com/courses/1225)

[3.0.4: Welcome to PyPoll](https://courses.bootcampspot.com/courses/1225)

Immersive Reader

**3.0.4**

# Welcome to PyPoll

[Previous](https://courses.bootcampspot.com/courses/1225/modules/items/497080)[Next](https://courses.bootcampspot.com/courses/1225/modules/items/497092)

© 2020 - 2022 Trilogy Education Services, a 2U, Inc. brand. All Rights Reserved.

[Bootcamp: UCD-VIRT-DATA-PT-03-2022-U-B-MW](https://courses.bootcampspot.com/courses/1225)

[3.1.1: The Command Line](https://courses.bootcampspot.com/courses/1225)

Immersive Reader

**3.1.1**

# The Command Line

**Tom's** manager, Seth, would like him to become familiar with the command line, which will be used to make updates to the GitHub repository, access local files and folders, and write and run Python programming scripts. Using the command line is a common practice among programmers, and you want to make a great impression. In this section, you'll get up to speed on the command line in order to help Tom with the election analysis.

Software developers and programmers typically use the command line to navigate and perform tasks on their computers. We'll be using the command line to access files and folders as well as update GitHub repositories, and starting the Python interpreter to write and run Python scripts.

How you access the command line depends on what operating system you're using. If you use macOS, you'll use the Terminal application to access the command line; if you have a computer running Windows, you'll use the Command Prompt. However, for cloning and updating GitHub repositories, you'll need to use Terminal for macOS or Git Bash for Windows.

Git Bash is an application for Microsoft Windows environments that installs Bash and Git on a Windows operating system. If you have a computer running Windows, we'll install Git Bash later. Bash is an acronym for "Bourne-again shell."  A **shell** is a terminal application, like Terminal on macOS, used to interface with an operating system through written commands.

**NOTE**

In this module, we'll be using the term **command line** to refer to both the Terminal application for macOS and Command Prompt for Windows.

Check out the macOS instructions below, or jump to the [Windows instructions](https://courses.bootcampspot.com/courses/1225/pages/3-dot-1-1-the-command-line?module_item_id=497092#windows).

## macOS

#### Opening the Terminal Application

On a Mac, the command line is accessed via the Terminal application. To open the Terminal application, complete the following steps:

1. Press Command+Space to open Spotlight Search.
2. Type "Terminal" into the search bar and press Enter.
3. Next, select the Terminal application to launch it. You can save Terminal to your dock for easy access, as we'll be using it often.

#### Using the Terminal Application

After we have launched the Terminal application using the previous instructions, we'll see an empty window. However, if you've logged in previously, you'll probably see the date and time of your most recent login. For example:

Last login: Thu May 2 11:27:18 on ttys001

Toms-MBP:~ tom$

At this point, the computer is waiting for further instruction. Let's break down what we see in this window.

The first line shows the details of the last login. The second line provides the name of the computer (in this case, Toms-MBP). The second part, ~ tom$, has three key parts:

1. The squiggly line, ~, is how the computer lets us know we are in the home directory.
2. The name "tom" is the name of the **home directory**. The home directory on a Mac is the little house that appears in the sidebar of the Finder window under Favorites.
3. The $ is what programmers call the **prompt**, which is where we will type commands.

**PRO TIP**

**NOTE**

Throughout this module, we'll be working on Tom's computer, since we're helping him with the election analysis. Therefore, images and code will show directory naming that is specific to Tom's computer. When you follow the steps on your computer, you'll see naming that corresponds to your specific machine.

#### Finding Your Home Directory

Every time you open the terminal application, it will open in the home directory. Using commands, we can find the terminal "path" to the home directory.

To find the path to your home directory (or any directory), enter the command pwd after the prompt, $. Then press Enter to run, or execute, the command.  pwd stands for "print working directory." The **working directory** is the current directory you are working in. For example, right now we are working in the home directory.

Toms-MBP:~ tom$ pwd

After we run the pwd command, the computer responds with /Users/tom on Tom's computer. This means that we're currently in the folder "tom," which is in the Users folder.

To find where /Users/tom is located on Tom's computer, we need to do the following:

1. Open the Finder window.
2. Locate "Devices." This may be listed under Favorites. For example, on Tom's computer, the name of the device is "Toms-MBP".
3. Find the Users folder. Click on that folder to find the "tom" folder. This is the folder we are in when we see Toms-MBP:~ tom$ in the command line.

You probably view files and folders on your computer by using the user interface and clicking on a folder name. With the command line, we can navigate to any folder on our computer by typing the appropriate commands.

The command to view files and folders in the terminal is ls, which means "list files." If we type ls after the prompt and press Enter, the terminal will print the folder names in alphabetical order from top to bottom and right to left. These are the folders in Tom's home directory:

Applications Downloads Pictures

Desktop Movies Public

Documents Music

#### Navigating with the Terminal Application

Let's say Tom wants to navigate to the Downloads folder. To do this, type cd Downloads after the prompt. The cd command means "change directory," and "Downloads" is the folder name. After typing the command, press Enter. The terminal returns the following:

Toms-MBP:~ Downloads tom$

If Tom wants to view folders on the Desktop, he would need to tell the computer to return to the home directory and then "change directory" to Desktop. (Remember, the Desktop folder is in the home directory, or the "tom" folder.)

If Tom's current working directory was Toms-MBP:~ Downloads tom$, then he would need to type cd .. after the prompt and press Enter. The two dots after cd tells the computer to go back one level, which, in our case, is the home directory. Next, he would type cd Desktop and press Enter. Now Tom would be in his Desktop directory, or folder.

An alternative to navigating to the Desktop folder for Tom would be for him to type cd ~ after the prompt. Remember, the "~" line means the home directory. It's like a magic command! No matter where you are in the command line, if you type cd ~, you'll be taken back to the home directory.

A technique used by more experienced programmers is to type cd ../Desktop after the prompt and then press Enter. This command allows you to go back one directory, or folder, level from the Downloads directory to the home directory, and then navigate to the Desktop folder. Let's break this command down in detail.

* The cd .. tells the computer to go back up one level.
* The forward slash "/" is a path separator.
* After the forward slash, we add the directory or folder, Desktop.

**CAUTION**

If you misspell the name of the directory when performing any of these commands, or type the name of a folder that is not in a directory, you will get an error that looks something like this:

-bash: cd: ../Documents: No such file or directory

If you misspell something on your first attempt, type "ls" and then press Enter to find the correct spelling.

#### Download and Install Git and Git LFS

In order to interact with and use GitHub, we need to download the Git software. There are several options to install Git on a Mac as indicated on the [Git download page (Links to an external site.)](https://git-scm.com/download/mac).

We will use the first option, brew install git.

If you already have Git installed skip to step 4 and 5.

First, to install Git using brew we'll need to install Homebrew.

1. Go to the [Homebrew (Links to an external site.)](https://brew.sh/) webpage and copy the script listed under "Install Homebrew" or copy the following script.

/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"

1. Paste the script into your command line and press enter.
2. After Homebrew has been installed type and run brew install git.
3. Next, check the version of Git by typing git --version.
   * The output should be git version 2.31.1 or greater.
4. If you have an older version of Git you can upgrade Git by typing and running brew upgrade git in the command line.

Finally, we'll install Git LFS (Large File Storage). Git LFS decreases the upload and download time for large files (>100MB) to and from GitHub.

1. To install Git LFS type and run brew install git-lfs in the command line.
2. Check the version of Git LFS by typing git-lfs --version.
   * The output should be git-lfs/2.13.3 or greater.

Once you've installed Git and Git LFS you are ready to interact with and use GitHub.

## Windows

#### Opening the Command Prompt

On a PC, the command line is accessed via the Command Prompt. If you are using Windows 8 or 8.1, click the search icon in the top right of your screen and type "Command Prompt." You will probably see the Command Prompt application appear before you finish typing the name. Click on the Command Prompt application to launch it.

If you are using Windows 10, type "Command Prompt" in the search bar. Click on the Command Prompt application to launch it.

#### Using the Command Prompt

Launch the Command Prompt on your machine. You can pin it to the taskbar for easy access, as we will be using it often. You'll likely have an empty window that looks like this:

At this point, the computer is awaiting further instructions. Let's take a closer look at this window; specifically, turn your attention to the line that says C:\Users\tom >.

Users\tom tells us the directory we're in. Currently, we're in the "tom" folder, which is located in the Users folder.

At any time, you can find the current directory with the command chdir. If you type this command after  >, the Command Prompt will look similar to this:

C:\Users\tom >chdir

C:\Users\tom

C:\Users\tom >

**PRO TIP**

To locate /Users/tom on our computer, open the Desktop folder and click Local Disk (C:). Then, open the Users folder, where we'll find the "tom" folder. This is the folder we're in when we see /Users/tom in the Command Prompt.

#### Navigating with the Command Prompt

You probably use your computer's user interface to view files and folders by double-clicking on a particular folder to see the contents. But you can also view all of your computer's folders and files using the command line, specifically, the dir command. Entering this command after the prompt will display a list of files and folders.

Type dir and press Enter. The terminal will return a list of folders along with a timestamp for the last time the folder or directory was accessed.

We can also use the command line to navigate to any folder on our computer simply by typing the appropriate command.

For example, if Tom is in the current directory, C:\Users\tom\Downloads and he wants to find which folders are on the Desktop, he needs to:

1. Tell the computer to return to the home directory.
2. "Change directory" to Desktop, because the Desktop folder is inside the "tom" folder.

Let's try this.

Type cd .. after the prompt and press Enter. The two dots tell the computer to go back one level, which, in our case, is C:\Users\tom. Then, type cd Desktop and press Enter. We are now in the Desktop directory.

As you become more proficient in using the Command Prompt, you will be able to combine commands instead of executing them separately. Here are two other methods for executing the commands to navigate to the Desktop folder from the Downloads folder.

1. At the prompt type cd ..\Desktop after the prompt. This will take you to the Desktop folder, which is back one directory. The backslash, "\", is a path separator. The directory name, Desktop, is included after the backslash.
2. If you know your folder structure, type cd C:\Users\tom\Desktop after the prompt to go directly to the Desktop folder.

#### Download Git Bash and Git LFS

When it comes to using GitHub on a Windows PC, we can't use the Command Prompt. Git uses commands for Linux or macOS, so these commands will not work with the Command Prompt. This is where Git Bash comes to the rescue!

To use Git on Window we will need to install the Git Bash, often called "Bash" for short. Programmers depend on this tool for **version control,** which is the process of logging the development of programs and applications. This comes in handy during collaborative programming, when teams of programmers change, add, and remove code throughout a project's directory. This process would be chaotic without Git.

Download the latest version of [Git Bash (Links to an external site.)](https://git-scm.com/downloads) for Windows and install it on your machine using the following steps:

**NOTE**

The version of Git will change often, but the link will always specify which version of Git you are downloading.

Next, we'll install Git LFS (Large File Storage). Git LFS decreases the upload and download time for large files (>100MB) to and from GitHub.

Launch Git Bash and type and run git lfs install at the prompt.

Once you've installed Git Bash and Git LFS you are ready to interact with and use GitHub.

#### Navigating with Git Bash

Navigating through directories with Git Bash is the same as navigating with the Command Prompt. A common command is pwd ("print working directory"), which is used to find our current location, or working directory. Type the command and press Enter. The output is the following:

/c/Users/tom

This is similar to using dir with the Command Prompt. However, we are not getting all the files in the directory. For this, use the ls command.

[Previous](https://courses.bootcampspot.com/courses/1225/modules/items/497085)[Next](https://courses.bootcampspot.com/courses/1225/modules/items/497097)

© 2020 - 2022 Trilogy Education Services, a 2U, Inc. brand. All Rights Reserved.

[Bootcamp: UCD-VIRT-DATA-PT-03-2022-U-B-MW](https://courses.bootcampspot.com/courses/1225)

[3.1.2: Install Python](https://courses.bootcampspot.com/courses/1225)

Immersive Reader

**3.1.2**

# Install Python

**You** will be using Python to write algorithms that will assist the confirmation and analysis of election results. Seth, Tom's manager, has informed you that you'll need to set up Python, and has given Tom the task of walking you through the installation process for both Windows and Mac computers.

Python is one of the most popular programming languages. You can think of Python as the newest power tool among programming languages. Although it is not new to the programming world, it is quickly becoming the "go-to" language for first-time programmers. The popularity of Python is due to the fact that it emulates the way a human thinks, which facilitates the process of writing code.

Once you download and install Python, Tom will guide you through writing a simple Python script or algorithm that can be executed, or run, to create a quick output to your computer screen. So, let's download and install Python—very soon you can start calling yourself a programmer!

Check out the macOS Python instructions below, or jump to the [Windows Python instructions](https://courses.bootcampspot.com/courses/1225/pages/3-dot-1-2-install-python?module_item_id=497097#python-for-windows).

## Python for macOS

#### Install Python

Python 2.7 comes preinstalled on Mac computers. This version of Python does not work with some of the updated programming dependencies that we will use (more on these dependencies later).

Therefore, let's install a newer version of Python, specifically, Python 3.7.6 which you can [download from the Python homepage (Links to an external site.)](https://www.python.org/downloads/). The following video will walk you through the installation process for macOS.

**NOTE**

Using a slightly older version of a software is typical in the software development industry. You need to install Python 3.7.6 because the coding presented in this module works with this version.

#### Use the Python Interpreter

The Python interpreter is a program that reads and executes Python code. When we start Python in the command line, it creates the interactive Python interpreter.

To use the Python interpreter in the command line, type python3 or python3 -i after the prompt, $, and then press Enter. The terminal window should now look something like this:

Python 3.7.6 (v3.7.6:43364a7ae0, Dec 18 2019, 14:18:50)

[Clang 6.0 (clang-600.0.57)] on darwin

Type "help", "copyright", "credits" or "license" for more information.

>>>

Let's break down what's happening here.

1. The first line provides the Python version that's running and the time and day of the creation of the version you are using.
2. There are tips you can use to get help and other information. All you have to do is type one of these words and press Enter.
3. The three chevrons, >>>, represent the Python prompt where you will type your code. Since Python is an interpretable programming language, all we need to do after the prompt is type any valid Python expression. Python will read the typed expression, evaluate it, and return the results in the line below the code you wrote.

Let's give it a try and write our first code or script that we can execute.

#### Write and Execute a Python Script

After the Python prompt, >>>, type print("Hello World"). Your Python terminal should look like this:

>>> print("Hello World")

Press Enter. You should see this output:

>>> print("Hello World")

Hello World

>>>

Notice print in the first line. The print() function is used to send whatever is between the parentheses of the print() function to screen. Inside the parentheses, we pass, or type, what we want to print. In this example, we typed Hello World. When we pressed Enter, Hello World was printed to the screen.

Let's pause to reflect what you just did: you wrote and executed your first Python script or algorithm. Congratulations!

For now, let's quit the Python interpreter. At the prompt, type exit() and press Enter. You will then see the terminal prompt.

## Python for Windows

#### Install Python

Now that you're familiar with the command line, it's time to install Python. In this module, we'll use Python 3.7.6 which you can [download from the Python homepage (Links to an external site.)](https://www.python.org/downloads/). The following video will walk you through the installation process for Windows.

**NOTE**

Using a slightly older version of a software is typical in the software development industry. You need to install Python 3.7.6 because the coding presented in this module works with this version.

Note that you can customize the installation, but we don't recommend this option unless you already have some experience with custom installing software. Then, check the "Add Python 3.7.6 to PATH" box.

**NOTE**

The "PATH" variable lists the directories that will be searched for executables when you type python in the command prompt.

By adding the path to the Python executable, you will be able to access the python.exe program by typing python, and you won't need to specify the full path to the program.

For Tom, the full path to the program would be C:\Users\tom\AppData\Local\Programs\Python\Python36.

If you don't check the box, then you will have to copy your path to the python.exe program, under \"Install Now," and type that everytime you want to open Python from the command line.

#### Use the Python Interpreter

The Python interpreter is a program that reads and executes Python code. When we start Python in the command line, it creates the interactive Python interpreter.

There are three ways to use the Python interpreter on Windows.

1. Open the Python 3.7.6 (64-bit) Command Prompt.
2. Open Windows Command Prompt and type python or python -i after the prompt,>, and Press Enter.
3. Open Git Bash and type python or python -i after the prompt, $, and Press Enter.

The choice for you depends on how comfortable you are starting a program, but the first option is probably the most straightforward.

When you use any of the methods to start the Python interpreter, the application window should look like this.

Python 3.7.6 (tags/v3.7.6:43364a7ae0, Dec 19 2019, 00:42:30) [MSC v.1916 64 bit (AMD64)] on win32

Type "help", "copyright", "credits" or "license" for more information.

>>>

Let's break down what's happening here.

1. The first line tells us the Python version that's running, the time and day of the creation of the version you are using, and the processor you are using on your computer.
2. There are tips you can use to get help and other information. All you have to do is type one of these words and press Enter.
3. The three chevrons, >>>, represent the Python prompt where you will type your code. Since Python is an interpretable programming language, all we need to do after the prompt is type any valid Python expression. Python will read the typed expression, evaluate it, and return the results in the line below the code you wrote.

Let's give it a try and write our first code or script that we can execute.

#### Write a Python Script

After the Python prompt, >>>, type print("Hello World"). Your Python terminal should look like this:

>>> print("Hello World")

Press Enter. You should see this output:

>>> print("Hello World")

Hello World

>>>

Notice print in the first line. The print() function is used to send whatever is between the parentheses of the print() function to screen. Inside the parentheses, we pass, or type, what we want to print. In this example, we typed Hello World. When we pressed Enter, Hello World was printed to the screen.

Let's pause to reflect what you just did: you wrote and executed your first Python script or algorithm. Congratulations!

[Previous](https://courses.bootcampspot.com/courses/1225/modules/items/497092)[Next](https://courses.bootcampspot.com/courses/1225/modules/items/497103)

© 2020 - 2022 Trilogy Education Services, a 2U, Inc. brand. All Rights Reserved.

[Bootcamp: UCD-VIRT-DATA-PT-03-2022-U-B-MW](https://courses.bootcampspot.com/courses/1225)

[3.1.3: Install Visual Studio Code](https://courses.bootcampspot.com/courses/1225)

Immersive Reader

**3.1.3**

# Install Visual Studio Code

**Tom** informs you that using the Python interpreter isn't the only way you'll be writing and executing Python scripts. As you write the script to perform the election audit, you will need to edit and save often. To help with this task, Tom suggests that you download one final piece of software: Visual Studio Code. Visual Studio Code, or VS Code, allows you to write and execute scripts, as well as edit and save scripts like you would a Microsoft Word or Excel file. To ensure you have the latest version, Tom is going to walk you through how to download and install VS Code.

Visual Studio Code, also called VS Code, is a free text editor that runs on macOS and Windows. VS Code is one of the better code editors currently available, and it makes it easy to work with a variety of different programming languages such as HTML, JavaScript, SQL, and, of course, Python.

**NOTE**

It's recommended that you use VS Code to complete this module.

There are two drawbacks when it comes to using the Python interpreter:

1. You can't save the code you write in the interpreter.
2. It's challenging to write scripts that are longer than a few lines.

This is where the VS Code outshines the Python interpreter.

To download VS Code, follow the instructions for your operating system.

## macOS

## Windows

[Previous](https://courses.bootcampspot.com/courses/1225/modules/items/497097)[Next](https://courses.bootcampspot.com/courses/1225/modules/items/497105)

© 2020 - 2022 Trilogy Education Services, a 2U, Inc. brand. All Rights Reserved.

[Bootcamp: UCD-VIRT-DATA-PT-03-2022-U-B-MW](https://courses.bootcampspot.com/courses/1225)

[3.1.4: Create a GitHub Repository](https://courses.bootcampspot.com/courses/1225)

Immersive Reader

**3.1.4**

# Create a GitHub Repository

**Before** you leave for the day, Tom thinks it would be a good idea to create a GitHub repository for the election analysis. Creating a GitHub repository will allow you to add the latest version of your project code on GitHub. This way, Tom and Seth can look over your progress on the project and provide assistance if you need it.

It's a common practice among programmers to store scripts in a location that other team members can view, comment on, and edit. Enter GitHub! We'll create a GitHub repository to help us organize, save, and share our code with our manager and other team members. Follow these steps:

[Previous](https://courses.bootcampspot.com/courses/1225/modules/items/497103)[Next](https://courses.bootcampspot.com/courses/1225/modules/items/497109)

© 2020 - 2022 Trilogy Education Services, a 2U, Inc. brand. All Rights Reserved.

[Bootcamp: UCD-VIRT-DATA-PT-03-2022-U-B-MW](https://courses.bootcampspot.com/courses/1225)

[3.1.5: Clone a GitHub Repository](https://courses.bootcampspot.com/courses/1225)

Immersive Reader

**3.1.5**

# Clone a GitHub Repository

**Nice** work creating the GitHub repository! Now Seth and Tom suggest that you reinforce your command line skills by copying, or cloning, the repository onto your computer.

Now that we've created the GitHub repository, we will copy, or clone, the repository and its contents to our computer. The git clone command creates a local copy of the GitHub folder structure, which will allow you to edit and run your code on your computer.  Follow these steps to clone the repository:

1. Navigate to the GitHub repository.
2. On the repository homepage, click the green "Clone or download" button in the top right corner.
3. When you see the "Clone with HTTPS" in the dropdown menu, copy the link to the clipboard.

**NOTE**

You can use either SSH or HTTPS with GitHub. We'll be using HTTPS because it is easier to clone and requires less setup. When you use HTTPS to clone a repository, you need to enter your GitHub username and password. This may also be a required step when you add files or folders to the repository from the command line.

A more secure method than entering your password is to generate a personal access token (PAT). Once generated from the GitHub website, you can copy your token and enter that instead of your password when prompted in the command line.

For more information about PATs, refer to the [GitHub documentation on Creating a PAT (Links to an external site.)](https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/creating-a-personal-access-token)

SSH is another secure way to download and upload files, but it can be trickier to set up. Once set up, all you need to do is enter your SSH password when you clone a repository. Your repository will be automatically associated with your SSH keys.

For more information about SSH, refer to the [GitHub documentation on SSH (Links to an external site.)](https://help.github.com/en/articles/connecting-to-github-with-ssh).

From here, the steps to access the cloned repository differ depending on your operating system.

## macOS

**REWIND**

Remember, to navigate to a folder in the command line, use the following commands:

* pwd: find your location
* cd: change the directory
* ls: list all the files and folders in the current directory

Once the repository has been cloned, you should see a folder on your computer with the same name as the repository.

## Windows

**REWIND**

Remember, to navigate to a folder in Git Bash, use the following commands:

* pwd: find your location
* cd: change the directory
* ls: list all the files and folders in the current directory

Once the repository has been cloned, you should see a folder on your computer with the same name as what is in the repository.

[Previous](https://courses.bootcampspot.com/courses/1225/modules/items/497105)[Next](https://courses.bootcampspot.com/courses/1225/modules/items/497118)

© 2020 - 2022 Trilogy Education Services, a 2U, Inc. brand. All Rights Reserved.

[Bootcamp: UCD-VIRT-DATA-PT-03-2022-U-B-MW](https://courses.bootcampspot.com/courses/1225)

[3.2.1: Create a Python File](https://courses.bootcampspot.com/courses/1225)

Immersive Reader

**3.2.1**

**Create a Python File**

**Now** that you have downloaded VS Code, Tom will walk you through a short tutorial on how to create and run Python files using VS Code. Using a text editor like VS Code is important because the Python scripts you will write for your analysis will be long and need to be edited often, but overall it will speed up your progress.

A big advantage of VS Code over the Python interpreter is that VS Code will let you know if you have syntax errors or misspellings. And, when we create longer scripts it can help with bracket-matching and code indentation, terms that we will go over in this module.

Let's get started with some basics of VS Code. First, we will walk through how to create a Python file.

1. Launch VS Code.
2. On the Start page, click "Open folder…"
3. Navigate to the Election\_Analysis folder that you created for this project.
4. When the folder is open in VS Code, click the "New File" icon.
5. In the blank space for the filename, type "Python\_practice.py." Press Enter to save the file.

**IMPORTANT**

If you do not add the .py extension to the Python file, you will not be able to run the file.

**NOTE**

To get started using VS Code, see the [official guide (Links to an external site.)](https://code.visualstudio.com/docs/introvideos/basics).

[Previous](https://courses.bootcampspot.com/courses/1225/modules/items/497109)[Next](https://courses.bootcampspot.com/courses/1225/modules/items/497123)

© 2020 - 2022 Trilogy Education Services, a 2U, Inc. brand. All Rights Reserved.

[Bootcamp: UCD-VIRT-DATA-PT-03-2022-U-B-MW](https://courses.bootcampspot.com/courses/1225)

[3.2.2: Execute Python Files](https://courses.bootcampspot.com/courses/1225)

Immersive Reader

**3.2.2**

# Execute Python Files

**Before** handing off your files to Tom and Seth, you will need to run, or execute, them to make sure the code is working. Tom is going to walk you through how to execute a Python file in VS Code. However, Seth thinks you should also be able to run Python files from the command line; he will be running your Python files from the command line and is only concerned about the final results. By running your files from the command line, you can make sure that their output is correct.

Knowing how to run Python files in both a text editor and from the command line will make you a more versatile programmer. Using a text editor is great for catching syntax errors while writing (or, as programmers like to say, "debugging your script"). You wouldn't want to submit a written assignment with spelling and grammatical errors that cause your writing to not make sense; this same principle applies to writing code.

When you create a script using an editor that returns the intended output, you can run that script from the command line any time by navigating to the directory where the file is located and using the appropriate commands. You already know how to navigate to different directories in the command line, so this process should feel familiar to you.

Let's get started and run your first Python script using VS Code.

Check out the macOS instructions below, or jump to the [Windows instructions](https://courses.bootcampspot.com/courses/1225/pages/3-dot-2-2-execute-python-files?module_item_id=497123#windows).

## macOS

Use the following instructions to learn how to run a Python file in VS Code as well as from the command line.

#### Run a Python File in VS Code

In the Python\_practice.py file in VS Code, type print("Hello World"). Your Python file should look like this:

The white dot next to the filename indicates that the file has been edited but has not been saved. If we want to execute this file, we need to save it. Save the file using the keyboard shortcut Command+S or by selecting "Save" from the File menu.

To run the Python file, we can either right-click on the filename or press the CTRL key and touchpad while selecting the filename in the Explorer. Then select "Run Python File in Terminal."

You can also click anywhere in Python\_practice.py and select "Run Python File in Terminal."

Both methods will open a window below your code. In that window, you will see output that looks similar to this:

Toms-MBP:Election\_Analysis tom$ /Users/tom/local/bin/python/Users/tom/Desktop/Class/Election\_Analysis/Python\_practice.py

Hello World

Toms-MBP:Election\_Analysis tom$

Before moving on, let's break down this output.

* The first line indicates the directory where the Python file is located.
* /Users/tom/local/bin/python is the command to run Python.
* /Users/tom/Desktop/Class/Election\_Analysis/Python\_practice.py is the location of the Python file that will be executed.
* Hello World is the result of the Python file, Python\_practice.py, being executed.
* Finally, the terminal prompt appears, waiting for the next command.

#### Run a Python File from the Command Line

To run a Python file in the Terminal app, we need to navigate to the folder that contains the file we want to run. Go to the Election\_Analysis folder.

**REWIND**

Remember, to navigate to a folder using the command line, use the following commands:

* pwd: find your location
* cd: change the directory
* ls: list all the files and folders in the current directory

The terminal window should look like this:

Toms-MBP:Election\_Analysis tom$

After the $ prompt, type python3 Python\_practice.py.

**IMPORTANT**

Typing python3 ensures that you'll use Python 3.7 and not the default Python 2.7 that comes preinstalled on your Mac.

After pressing Enter, the terminal window should look like this:

Toms-MBP:Election\_Analysis tom$ python3 Python\_practice.py

Hello World

Toms-MBP:Election\_Analysis tom$

Your output should be Hello World.

## Windows

Use the following instructions to learn how to run a Python file in VS Code as well as in the Python Command Prompt.

#### Run a Python File in VS Code

In Python\_practice.py in VS Code, type print("Hello World"). Your Python file should look like this:

The white dot next to the filename indicates that the file has been edited but has not been saved. If we want to execute this file, we need to save it. Save the file using the keyboard shortcut CTRL+S or you by selecting "Save" from the File menu.

To run the Python file, we can click on the filename in the Explorer and select "Run Python File in Terminal."

We can also click anywhere in Python\_practice.py and select "Run Python File in Terminal."

Both methods will open a window below the code. In this window, you will see output that looks similar to this:

C:\Users\tom\Desktop\Election\_Analysis>C:/Users/Tom/Python36/python.exe c:/Users/tom/Desktop/Election\_Analysis/Python\_practice.py

Hello World

C:\Users\tom\Desktop\Election\_Analysis>

Let's break down this output:

* The first line indicates the directory where your Python file is located.
* C:/Users/tom/Python36/python.exe is the command to run Python.
* c:/Users/tom/Desktop/Election\_Analysis/Python\_practice.py is the location of the Python file that will be executed.
* Hello World is the result of the Python file being executed.
* Finally, the terminal prompt appears, waiting for the next command.

#### Run a Python File in the Python Command Prompt

To run a Python file in the Command Prompt, we first need to navigate to the folder that contains the file we want to run. Launch the Command Prompt. The terminal window should look like this:

C:\Users\tom>

Navigate to the Election\_Analysis folder in the Command Prompt.

**REWIND**

Remember, to navigate to a folder using the command line you can use the following commands.

* pwd: find your location
* cd: change the directory
* ls: list all the files and folders in the current directory

The Command Prompt window should look similar to this:

C:\Users\tom\Desktop\Election\_Analysis>

After the > prompt, type python Python\_practice.py. Press Enter. Now the terminal window should look like the following:

C:\Users\tom\Desktop\Election\_Analysis>python Python\_practice.py

Hello World

C:\Users\tom\Desktop\Election\_Analysis>

The output should be Hello World.

#### Run a Python File in Git Bash

To run a Python file in Git Bash, we first need to navigate to the folder that contains the file we want to run. Launch Git Bash. The Git Bash window should look like this:

tom@TOMD249 MINGW64 ~

$

Now navigate to the Election\_Analysis folder in the Command Prompt.

The Git Bash window should look like this:

/c/Users/tom/

tom@TOMD249 MINGW64 ~/Desktop/Election\_Analysis

$

After the $ prompt, type python Python\_practice.py. Press Enter. The terminal window should look similar to the following:

/c/Users/tom/

tom@TOMD249 MINGW64 ~/Desktop/Election\_Analysis

$ python Python\_practice.py

Hello World

tom@TOMD249 MINGW64 ~/Desktop/Election\_Analysis

$

The output should be Hello World.

[Previous](https://courses.bootcampspot.com/courses/1225/modules/items/497118)[Next](https://courses.bootcampspot.com/courses/1225/modules/items/497127)

© 2020 - 2022 Trilogy Education Services, a 2U, Inc. brand. All Rights Reserved.

[Bootcamp: UCD-VIRT-DATA-PT-03-2022-U-B-MW](https://courses.bootcampspot.com/courses/1225)

[3.2.3: Data Types](https://courses.bootcampspot.com/courses/1225)

Immersive Reader

**3.2.3**

# Data Types

**Now** that you are familiar with opening the Python interpreter, Tom is going to walk you through some of the basics of the Python language. This way, you'll be up-to-speed on the skills needed to perform the election audit. The first thing Tom will do is go over the data types you're likely to encounter. Knowing how to identify the data types you are working with is key, because only certain data types can be used for calculations. In the election analysis, you will be performing mathematical calculations as well as getting non-mathematical data from a file.

There are a variety of data types in Python, some of which you may encounter in the election dataset. Because you need to be prepared to handle different types of data, let's discuss a few key data types: integers, floating-point decimal numbers, strings, and Boolean values.

## Integers

**Integers** are used to perform mathematical calculations. The length of an integer is constrained by the amount of RAM memory on your computer. For example, a computer with 8 GB of RAM will have 8,000,000,000 bytes. The integer 10,000,000,000,000,000,000, or 10 quintillion, will only take up 36 bytes of memory.

**REWIND**

 An integer is a positive or negative whole number. Whole numbers between –32,768 and 32,767 are stored in 16 bits, and whole numbers between –2,147,483,648 and 2,147,483,647 are stored in 32 bits.

Let's determine the data type of a given integer using the command line. To get started, open the command line and activate the Python interpreter.

**NOTE**

To use Python in the command line:

* If you're on a Mac, type python3.
* If you're using Windows, open the Python 3.7 (64-bit) Command Prompt.

The type() function is used to determine data type. Inside this function, add the integer or other value to determine the data type. For example, let's say we want to find the data type of the integer 3. Type type(3) after the Python interpreter prompt, as shown, and then press Enter.

>>> type(3)

The output shows that the data type for the number 3 is an integer, as denoted by the 'int'.

>>> type(3)

<class 'int'>

Note that when typing integer values greater than 999, a thousands separator, or comma, should **not** be used. Inserting commas can make the whole number you thought you were typing something different. For example, the variable "ballots" is equal to 1,341.

>>> ballots = 1,341

>>> ballots

(1, 341)

>>>

When we type "ballots" in the next line, the output is not 1,341, but 1 and 341. This is known as the Python data type called a **tuple** (which we'll encounter later), containing two numbers.

We can confirm that "ballots" is a tuple by determining its data type.

>>> type(ballots)

<class 'tuple'>

>>>

**IMPORTANT**

When typing integers greater than 999 in Python, do not use a thousands separator (comma).

## Floating-Point Decimal Numbers

Like integers, **floating-point decimal numbers** are used to perform mathematical calculations. Floating-point decimal numbers specify numbers that have a decimal point, like 73.81.

When we type type(73.81) and press Enter, the output shows that this is a floating-point decimal number, as indicated by "float."

>>> type(73.81)

<class 'float'>

## Strings

**String** variables can be text or numbers wrapped by either single or double quotes, or delimiters. A **delimiter** is any nonalphabetical characters used to specify the boundary between plain text or other numbers, like the single or double quotes, or opening and closing parentheses. All characters between the quotes are part of the string; for example, 'Hello World' or "Hello World".

To determine the data type of "Hello World", type the following:

type("Hello World")

Press Enter. The output shows that this is a string, 'str'.

>>> type("Hello World")

<class 'str'>

Sometimes, you may come across **empty strings**, which contain no text or numbers between the delimiters. A string that is empty is written as '', or "". Because there are opening and closing single or double quotes, the data type is still a string.

**SKILL DRILL**

Determine the data type of an empty string.

End of text box.

Unlike integers and floating-point decimal numbers, strings cannot be used to perform mathematical calculations. Also, it's important to note that if an integer or a floating-point decimal number is wrapped in quotes, it's considered to be a string, not an integer or decimal number. Later in this module, we'll go over how to change a data type from a string to an integer or floating-point decimal number.

## Boolean Values

**Boolean** data types can have one of two values: true or false. If we pass True in the type() function, we'll get the data type 'bool'.

**IMPORTANT**

To determine the data type of Boolean values, "True" or "False" must be capitalized when written in code.

>>> type(True)

<class 'bool'>

To summarize, here are four data types that we encountered in this module.

|  |  |
| --- | --- |
| **Data Type** | **Python Classification** |
| Integers | <class 'int'> |
| Float point numbers | <class 'float'> |
| Strings | <class 'str'> |
| Boolean | <class 'bool'> |

**NOTE**

For more information, read the [documentation on Python data types](https://docs.python.org/3.7/library/stdtypes.html#numeric-types-int-float-complex).

## Creating Variables in Python

Variables are essentially a way to store different data types to be used later. A **variable** is a name that represents a value that is stored in the computer's memory. A variable can be an integer, floating-point decimal number, string, or Boolean value. In Python, you can declare a variable by typing a variable name followed by an equals sign.

**REWIND**

After a variable is declared, it is assigned a value. This is done by referencing the variable by name and setting its value with an equals sign.

Here are a few examples of declared variables in Python.

num\_candidates = 3

winning\_percentage = 73.81

candidate = "Diane"

won\_election = True

## Naming Conventions for Python Variables

Variable names in Python can be any length and can consist of the following:

* Uppercase letters (A–Z)
* Lowercase letters (a–z)
* Digits (0–9)
* Underscores (\_).

When considering a variable name, keep the following in mind:

* The first character of a variable name cannot be a digit.
* Choose a variable name that reflects what variable will reference and can be understood by someone reading your code. For example, using x = "Diane" for a candidate's name is not a good choice; instead, use candidate\_name = "Diane".
* Variables are case-sensitive.
* Shorter names are better.
* Variables must be in **snake case,** which is when words are separated by underscores. For example: candidate\_name.

One final consideration: the Python language has a small set of **keywords**, or reserved words that have special language functionality. No variable can have the same name as a keyword. These reserved words in Python are listed below.

|  |  |  |  |
| --- | --- | --- | --- |
| **False** | **def** | **if** | **raise** |
| None | del | import | return |
| True | elif | in | try |
| and | else | is | while |
| as | except | lambda | with |
| assert | finally | nonlocal | yield |
| break | for | not |  |
| class | from | or |  |
| continue | global | pass |  |

**NOTE**

You can also get this list if you type help("keywords") in the command line.

When you follow these rules, you can create variables with any name that you like. Remember, it's recommended that you choose short, simple names that accurately reflect what the variable represents.

[Previous](https://courses.bootcampspot.com/courses/1225/modules/items/497123)[Next](https://courses.bootcampspot.com/courses/1225/modules/items/497131)

© 2020 - 2022 Trilogy Education Services, a 2U, Inc. brand. All Rights Reserved.

[Bootcamp: UCD-VIRT-DATA-PT-03-2022-U-B-MW](https://courses.bootcampspot.com/courses/1225)

[3.2.4: Perform Calculations Using Python](https://courses.bootcampspot.com/courses/1225)

Immersive Reader

**3.2.4**

# Perform Calculations Using Python

**Now** that you are familiar with integers and floating-point decimal numbers, Tom is going to walk you through the process of performing simple and complex mathematical calculations in Python. Some mathematical operations will look familiar to you, but there are some operations that be new to you if don't have prior Python programming experience. Don't worry—you'll get the hang of it! Learning how to perform these mathematical operations is essential in programming, and Seth and Tom want to make sure you are comfortable using them in the election audit.

Many real-world coding programs require mathematical calculations like addition, subtraction, multiplication, and division. To perform these calculations in Python, several **arithmetic operators** are used. The following table lists these arithmetic operators, their meanings, and how they are used.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Meaning** | **Use** |
| + | Adds two numbers. | x + y |
| – | Subtracts one number from another. | x – y |
| \* | Multiplies two numbers. | x \* y |
| / | Divides one number by another. This always results in a floating-point decimal number. | x / y |
| % | The “%” is known as the **modulus**. When used in place of “/” it will divide one number by another, and return the remainder of the division. | x % y  (remainder of x/y) |
| // | Divides one number by another and returns an integer. This is known as **floor division**. | x // y |
| \*\* | Raises a number to a power. | x\*\*y  (x to the power y) |

## Order of Precedence

When you need to perform more complex mathematical expressions that include any combination of division, multiplication, addition, and subtraction, you must follow the **order of precedence (operations)** for arithmetic operators. The order of precedence in Python, which follows the same guidelines as in mathematics, is:

1. Operations enclosed in parentheses are performed first.
2. Exponentiations (i.e., raising a number to a power) are performed next.
3. Multiplication and division operations are performed from left to right: \*, /, //, and %.
4. Finally, addition and subtraction operations are performed from left to right.

**SKILL DRILL**

In the Python interpreter, evaluate the following mathematical expressions:

1. 5 + 2 \* 3
2. 8 // 5 - 3
3. 8 + 22 \* 2 - 4
4. 16 - 3 / 2 + 7 - 1
5. 3 \*\* 3 % 5
6. 5 + 9 \* 3 / 2 - 4

End of text box.

## Grouping with Parentheses

In a mathematical calculation, you can group expressions in parentheses to indicate that those expressions should be performed before operations there are **not** in parentheses.

Treat these calculations just like algebraic expressions: if parentheses are enclosed within parentheses, work from the inside out.

**SKILL DRILL**

In the Python interpreter, evaluate the following mathematical expressions and compare them with the answers in the previous exercise.

1. (5 + 2) \* 3
2. (8 // 5) - 3
3. 8 + (22 \* (2 - 4))
4. 16 - 3 / (2 + 7) - 1
5. 3 \*\* (3 % 5)
6. Finally, compare the following: 5 + (9 \* 3 / 2 - 4) and 5 + (9 \* 3 / (2 - 4))

End of text box.

[Previous](https://courses.bootcampspot.com/courses/1225/modules/items/497127)[Next](https://courses.bootcampspot.com/courses/1225/modules/items/497134)

© 2020 - 2022 Trilogy Education Services, a 2U, Inc. brand. All Rights Reserved.

[Bootcamp: UCD-VIRT-DATA-PT-03-2022-U-B-MW](https://courses.bootcampspot.com/courses/1225)

[3.2.5: Data Structures: Lists](https://courses.bootcampspot.com/courses/1225)

Immersive Reader

**3.2.5**

# Data Structures: Lists

**Now** that you have been exposed to Python data types, Seth and Tom want you to learn how data types are stored and accessed. In the election audit, you may need to store or access data in a variety of formats, such as lists, tuples, and dictionaries. Accessing the data in these data structures as well as storing data in new files are common tasks that data analysts perform every day.

One of the basic data structures in Python is a list. A **list** is an array that contains multiple data items, like the following list of counties.

The items in a list can be data types such as integers, floating-point decimals, strings, and Boolean values, as well as other data structures like lists, tuples, and dictionaries. (We'll cover tuples and dictionaries later in this module.)

**IMPORTANT**

When it comes to list, remember these three properties.

1. We can use **indexing** and **slicing** to retrieve specific items from the list.
2. We can add or remove items from a list, which makes lists a **dynamic** data structure.
3. We can change the contents in a list. For example, we can change "Jefferson" to "El Paso." This means lists are **mutable:** we can change one or more items in a list to something else.

Let's look at these three features in action.

Activate the Python interpreter. We'll first declare a list variable counties by typing the following code and then pressing Enter.

>>> counties = ["Arapahoe","Denver","Jefferson"]

**NOTE**

When we want to add items to a list but the list has not been declared, we must first declare an empty list. An **empty list** can be declared with the following syntax: my\_list = [ ]. Alternatively, you can use the built-in function list() to create an empty list: my\_list = list().

To confirm that this list is declared, type counties on the next line and press Enter. The output should be the following:

>>> counties

['Arapahoe', 'Denver', 'Jefferson']

## Index Lists

In Python, we access items in a list using indexing.

**REWIND**

An **index** of a variable is its position in the array. Here are some general rules for indexing:

1. Each item in a list has an index that specifies its position in the list.
2. Indexing starts at 0. Therefore, the index of the first item is 0, the index of the second number is 1, and so on.
3. Because indexing begins at 0, the index of the last item in a list is 1 less than the number of items in the list.

To get the **first** item in the counties list, type the following in the command line:

>>> counties[0]

Press Enter to get the output, which is 'Arapahoe'.

>>> counties[0]

'Arapahoe'

You can also use the print statement to print the county to the screen, like this:

>>> print(counties[2])

Jefferson

**Negative indexes** are used to identify a list item's position relative to the end of the list. For example, to find the **last** item in the counties list, we would enter the following:

>>> print(counties[-1])

Jefferson

To get the **second-to-last** item in a list, we would type counties[-2], and so forth.

## Find the Length of a List

To get the total number of items in a list, we use the len() function and then add the list inside the parentheses, like this: len(counties). When we execute this script, the output will be the number of items in the list, which is 3.

>>> len(counties)

3

## Slice Lists

Sometimes we'll need to retrieve certain items from the list. To do this, we can use the index values to slice a list. **Slicing** is used to get specific items from a list. The format for slicing a list is as follows: list[start : end].

Let's break down how slicing works.

1. The start refers to the index of the first item in the slice.
2. The end is the index marking the end of the slice.
3. The expression list[start : end] returns a list containing a copy of the items in the list from the starting index value up to, but not including, the ending index value.

For example, to find the first and second items from the counties list, we type counties[0:2], not counties[0:1].

The output of counties[0:2] will be Arapahoe and Denver, the first and second items from the counties list.

>>> counties[0:2]

['Arapahoe', 'Denver']

The output of counties[0:1] will be Arapahoe, the first item from the counties list.

>>> counties[0:1]

['Arapahoe']

Alternatively, you can use counties[:2] to get the first and second items from the counties list.

>>> counties[:2]

['Arapahoe', 'Denver']

With counties[:2], the beginning index is omitted with [:, so the slice contains the elements starting from 0 and ending at 2.

Another option to get Denver and Jefferson counties in a list is to use counties[1:]. Here, the ending index is omitted with :]; the slice contains the elements starting at 1 and ending at length of the list, or 3, since there are only three items in the counties list.

>>> counties[1:]

['Denver', 'Jefferson']

## Add Items to a List

Items can be added to an empty list or a list that already exists by using the append() function and the syntax list.append(). In the parentheses, add the data you want, whether integers, floats, strings, or another data type or data structure. For example, let's add a fourth county, El Paso, to the counties list:

>>> counties.append("El Paso")

When you press Enter, you will notice there is no output—but that doesn't mean nothing happened. To check, type counties and press Enter. The output prints the list with El Paso added to the list as the last item.

>>> counties

['Arapahoe', 'Denver', 'Jefferson', 'El Paso']

**NOTE**

Using the append() function on a list will always add the new item at the end of the list.

To specify where in a list to add a new item, select the location with an index by using the following syntax, list.insert(index, obj).

Here, index represents where we would like the new item to be placed, and obj represents the item.

Let's look at an example. We'll add another instance of "El Paso" at index 2, which is the third position in the counties list, to the counties from the previous example.

When we execute this code and print the counties list, the output is the following:

>>> counties.insert(2, "El Paso")

>>> counties

['Arapahoe', 'Denver', 'El Paso', 'Jefferson', 'El Paso']

To remove an instance of "El Paso" from our list, we'll append the .remove method and specify the list item we're removing.

In the example below, .remove("El Paso") was appended to our counties variable. In the output, the only instance of "El Paso" remaining is located in the third index.

>>> counties.remove("El Paso")

>>> counties

['Arapahoe', 'Denver', 'Jefferson', 'El Paso']

>>>

Another method that can be used to remove items from a list is the pop() method. The pop() method removes the item at a given index from the list and then returns the removed item. This is a good way to check which item was removed.

To remove the last instance of "El Paso" using the pop() method, enter counties.pop(3). "El Paso" is the fourth item in the list, which means its index is 3.

When we execute this code, the output will look like this:

>>> counties.pop(3)

'El Paso'

>>> counties

['Arapahoe', 'Denver', 'Jefferson']

>>>

## Change an Element in a List

Python lists are mutable, which means we can change the elements inside the list. To change an item in a list, use the syntax list[index] on the left side of the equals sign; on the right side, you assign the index a new data value. The new value can be an integer, floating-point decimal number, string, Boolean value, another list, tuple, or dictionary.

If you want to change Jefferson county to El Paso, you would type the following: counties[2] = "El Paso".

When you run the code, "Jefferson" is replaced by "El Paso."

>>> counties[2] = "El Paso"

>>> counties

['Arapahoe', 'Denver', 'El Paso']

>>>

**NOTE**

For more information, see the [documentation on built-in Python functions like list()](https://docs.python.org/3.7/library/functions.html).

[Previous](https://courses.bootcampspot.com/courses/1225/modules/items/497131)[Next](https://courses.bootcampspot.com/courses/1225/modules/items/497138)

© 2020 - 2022 Trilogy Education Services, a 2U, Inc. brand. All Rights Reserved.

[Bootcamp: UCD-VIRT-DATA-PT-03-2022-U-B-MW](https://courses.bootcampspot.com/courses/1225)

[3.2.6: Data Structures: Tuples](https://courses.bootcampspot.com/courses/1225)

Immersive Reader

**3.2.6**

# Data Structures: Tuples

**Storing** and accessing data in lists can be performed in different ways depending on what data you want to store or retrieve. Now Tom and Seth think you can move on to learning how to manipulate and use tuples. Like lists, tuples can be used to store data; however, you can only retrieve data from tuples. Tom is going to walk you through how to retrieve data from tuples.

**Tuples** are similar to lists in Python, with a major exception: once you create a tuple, it cannot be changed. This means tuples are **immutable**: we can't add or remove items from them.

**IMPORTANT**

You can create an empty tuple with the following syntax:

my\_tuple = ( )

Alternatively, you can use the built-in tuple() method:

my\_tuple = tuple()

Here's a tuple that contains a list of counties:

counties\_tuple = ("Arapahoe","Denver","Jefferson")

Like lists, we can find the length of tuples as well as find items in the tuple by index.

To get the length of a tuple, use the same format for getting the length of a list: len(counties\_tuple).

>>> len(counties\_tuple)

3

To get an item from a tuple, we can apply indexing and slicing, using square brackets after the tuple, just like with lists.

In the Python interpreter, type counties\_tuple[1] and press Enter. The output is Denver.

>>> counties\_tuple[1]

'Denver'

**NOTE**

For more information, see the [documentation on built-in Python functions like tuple() (Links to an external site.)](https://docs.python.org/3.7/library/functions.html).

© 2020 - 2022 Trilogy Education Services, a 2U, Inc. brand. All Rights Reserved.

[Bootcamp: UCD-VIRT-DATA-PT-03-2022-U-B-MW](https://courses.bootcampspot.com/courses/1225)

[3.2.7: Data Structures: Dictionaries](https://courses.bootcampspot.com/courses/1225)

Immersive Reader

**3.2.7**

# Data Structures: Dictionaries

**Now** that you're familiar with two common data structures, lists and tuples, Tom would like you to learn how data is stored and retrieved from a third data structure: dictionaries.  Real-world data is stored in dictionaries rather than lists or tuples. There are multiple ways to retrieve data from a dictionary, and Tom wants to make sure you are familiar with the different methods available. Some may come in handy for the elections analysis, while other methods you may need to use in upcoming projects.

A **dictionary** is an object that stores a collection of data. An object in Python and other computer programming languages is data stored in the computer's memory.

A Python dictionary has a **key** and a **value**, or **key-value** **pairs**. Very similar to a dictionary that contains definitions, the words in the dictionary would be considered the keys, and the definitions of those words would be the values.

Key-value pairs are enclosed in a set of curly braces, {}. The syntax for a dictionary is the key followed by a colon, which is followed by a value: {key:value}.

When we want to find a definition in a regular dictionary, we look up the word. Likewise, in Python, when we want to get a specific value from the dictionary, we look up the key.

If there's more than one element or key-value pair in the dictionary they are separated by commas, like this: {key1:value1, key2:value2}.

There are two key rules for dictionaries:

1. Values in a dictionary can be objects of any type: integers, floating-point decimals, strings, Boolean values, datetime values, and lists.
2. Keys must be immutable objects, like integers, floating-point decimals, or strings. Keys cannot be lists or any other type of mutable object.

## Create a Dictionary

During the data analysis process, you may be given a dictionary and have to retrieve data from the dictionary. Occasionally, you may need to initialize a dictionary in which you will store data as you collect it from other sources, like lists, tuples, or CSV files (as we will encounter later in this module).

To initialize or create an empty dictionary, we use the following syntax: my\_dictionary = {}. Or you can create a dictionary with the built-in Python dict() method, my\_dictionary = dict().

Let's create a dictionary with the counties as keys and the number of registered voters of each county as values.

Create the empty counties dictionary by typing the following code in the Python interpreter and pressing Enter.

>>> counties\_dict = {}

Next, add the county "Arapahoe" to the dictionary as the key and the number of registered voters for Arapahoe as the values for this key.

**IMPORTANT**

The standard format for creating a key in a dictionary is to place the key between single or double quotes and inside brackets.

>>> counties\_dict["Arapahoe"] = 422829

Let's see how this looks. Type counties\_dict on the next line and press Enter.

>>> counties\_dict

{'Arapahoe': 422829}

Repeat this process two more times to add the counties "Denver" and "Jefferson" with their respective number of registered voters.

>>> counties\_dict["Denver"] = 463353

>>> counties\_dict["Jefferson"] = 432438

When we enter counties\_dict and press Enter, the output in the interpreter should look like this:

>>> counties\_dict

{'Arapahoe': 422829, 'Denver': 463353, 'Jefferson': 432438}

Congratulations on creating a dictionary in Python!

**REWIND**

Remember, when typing integers in Python, do not use the thousands separator (comma) in the number.

## Get the Length of a Dictionary

Similar to lists and tuples, the length of a dictionary can be found by using the len() function. To get the length of counties\_dict, run the following code:

>>> len(counties\_dict)

3

>>>

## Get All Keys and Values

To get all the keys and values printed to the screen, simply print the dictionary name. Or, we can use the items() method on the dictionary. This will return a list of tuples where the first element in each tuple is the key of the dictionary, and the second element in each tuple is the value corresponding to that key.

If we add the items() method to the end of counties\_dict, we'll get this output:

>>> counties\_dict.items()

dict\_items([('Arapahoe', 422829), ('Denver', 463353), ('Jefferson', 432438)])

>>>

In the output, the information inside the dict\_items([]) is what is known as a **view object**. A view object will give us a snapshot of what is in the dictionary. In this output, we can see each key and their respective values.

**NOTE**

You cannot use list indexing with the items() method.

## Get All the Keys

To get only the keys from a dictionary, add the keys() method at the end of the dictionary, like this:

>>> counties\_dict.keys()

dict\_keys(['Arapahoe', 'Denver', 'Jefferson'])

>>>

The keys() method will return a view object that contains the keys of the dictionary as a list.

## Get All the Values

To retrieve only the values from a dictionary, add the values() method to the end of the dictionary, like this:

>>> counties\_dict.values()

dict\_values([422829, 463353, 432438])

>>>

Just like the items() and keys() methods, the values() method will return a view object that contains the values of the dictionary as a list.

## Get a Specific Value

There are two methods you can use to get a specific value from a dictionary.

The first method is to use the get() method. With the get() method, we pass a key inside the parentheses to "get" the value of that key.

Let's "get" the value or the number of registered voters, in Denver County. Type the following code and press Enter.

>>> counties\_dict.get("Denver")

463353

To get the number of registered voters in Arapahoe County, we can also wrap the key in brackets using this format: dictionary\_name[key].

When we enter a key to retrieve its value, we must type the key within single or double-quotes. Without quotes, we will get the following NameError.

>>> counties\_dict[Arapahoe]

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

NameError: name 'Arapahoe' is not defined

Using single or double quotes will satisfy the requirement.

>>> counties\_dict['Arapahoe']

422829

>>> counties\_dict["Arapahoe"]

422829

## Lists of Dictionaries

Sometimes Python dictionaries have the same keys associated with different values, which are written in this format:

[{key1:value1, key2:value2}, {key1:value3, key2:value4}].

This is referred to as a **list of dictionaries** because each dictionary, {}, is wrapped in brackets.

Let's create a list of dictionaries where the keys are "county" and "registered\_voters," and each county and its corresponding registered voters are the values for those keys.

First, create an empty list called voting\_data.

>>> voting\_data = []

Then add, or append, each dictionary to the voting\_data list with the following code and press Enter:

>>> voting\_data.append({"county":"Arapahoe", "registered\_voters": 422829})

>>> voting\_data.append({"county":"Denver", "registered\_voters": 463353})

>>> voting\_data.append({"county":"Jefferson", "registered\_voters": 432438})

When we type voting\_data and press Enter, we get the following output:

>>> voting\_data

[{'county': 'Arapahoe', 'registered\_voters': 422829}, {'county': 'Denver', 'registered\_voters': 463353}, {'county': 'Jefferson', 'registered\_voters': 432438}

Take a look at this code. This is a **list of dictionaries** because each element in the list is a dictionary. The keys are "county" and "registered\_voters."

The format of a CSV file is like a list of dictionaries, where the headers (column names) "county" and "registered\_voters" are the keys, and the values are the data in the rows. If you added this data to a CSV file it would look like this.

We can use list methods to:

1. Get the length of the voting\_data list of dictionaries.
2. Use indexing and slicing to get one or more dictionaries.
3. Use the append(), insert(), and remove() methods to add and remove one or more dictionaries.
4. Change a value for one of the keys in the list of dictionaries.

**NOTE**

For more information, refer to the documentation on the following topics:

* [Dictionary keys (Links to an external site.)](https://docs.python.org/3.7/faq/design.html#why-must-dictionary-keys-be-immutable)
* [View objects (Links to an external site.)](https://docs.python.org/3.7/library/stdtypes.html#dictionary-view-objects)
* [Built-in Python functions such as dict() (Links to an external site.)](https://docs.python.org/3.7/library/functions.html)

[Previous](https://courses.bootcampspot.com/courses/1225/modules/items/497138)[Next](https://courses.bootcampspot.com/courses/1225/modules/items/497146)

© 2020 - 2022 Trilogy Education Services, a 2U, Inc. brand. All Rights Reserved.

[Bootcamp: UCD-VIRT-DATA-PT-03-2022-U-B-MW](https://courses.bootcampspot.com/courses/1225)

[3.2.8: Decision Statements](https://courses.bootcampspot.com/courses/1225)

Immersive Reader

**3.2.8**

# Decision Statements

**Now** that you are familiar with data storage and retrieval, Seth and Tom would like you to learn how to retrieve data based on whether a condition is true or false. Retrieving data based on a condition is known as a decision statement. Decision statements are widely used by programmers because they often need to retrieve data from a large dataset that meets certain criteria. Tom is going to walk you through the structure of a simple decision statement and demonstrate how to create and use them. Then, he'll show you how to create and implement more complex decision statements.

Some programming problems can be solved by performing a series of ordered steps. For example, if you need to create an algorithm that calculates the percentage of votes a candidate receives in an election, you might write a simple algorithm like this:

# How many votes did you get?

my\_votes = int(input("How many votes did you get in the election? "))

# Total votes in the election

total\_votes = int(input("What is the total votes in the election? "))

# Calculate the percentage of votes you received.

percentage\_votes = (my\_votes / total\_votes) \* 100

print("I received " + str(percentage\_votes)+"% of the total votes.")

The steps represented in this algorithm are:

1. Declare the "my\_votes" variable equal to an input function. The input function will prompt the user to type an amount, such as 200.
2. Wrap the input function with the int() method. When we use an input function, the data type of the user input defaults to a string. The int() method converts the user input value to an integer, which is necessary to perform the calculation for the percentage of votes. If we want the user to enter a floating-point decimal number, then we would use the float() method.
3. Calculate the percentage of votes by dividing the users' votes by the total votes, and then multiplying by 100 using the multiplication operator, the asterisk.
4. Create a print statement that tells the percentage of votes. The percentage\_votes must be converted from int to a string data type using str(), which is necessary in order to print the percentage of votes in the sentence.

**NOTE**

The input function asks a user for text input. Once the user inputs text and presses Enter, the program will resume.

However, if you needed to create an algorithm that determines if a candidate won the election based on the majority of the votes, which is equal or greater than 50.1%, then the above program would require a control structure to execute a set of statements only under certain conditions. This type of program is called a **decision statement** or decision structure.

We'll cover two types of decision statements: if statements and if-else statements.

## If Statements

In a simple decision structure like an if statement, if a condition is met (i.e., is true), then an action is performed. If the condition is not met (i.e., is false), then the action is not performed. This can be illustrated with the following flowchart:

In Python, the general format for the if statement is to write a single alternative decision as follows:

if condition:

statement 1

statement 2

**REWIND**

Remember, if statements tell the computer that certain lines of code should only run under certain conditions. The if statement checks if a condition is true. If it's true, then a block of code below it will run.

Here are the general rules when using if statements in Python:

* If statements begin with the word "if," followed by a condition. The condition can be an expression like if len(counties) > 2:, which will be evaluated as either true or false. At the end of the condition is a colon.
* The next line contains a block of statements. All statements in a block must be consistently indented. This indentation is required because the Python interpreter or VS Code editor uses it to determine where the block begins and ends. If you do not indent, your code will not be executed.
* If the statement is not met (i.e., is false), then the statements in the block are skipped.

Let's practice using the if statement on our counties list to determine if the second county in the list is Denver. If so, then we will print "Denver" to the screen.

Start by opening VS Code and creating a new file called Python\_practice.py in the Election\_Analysis folder.

**REWIND**

Add the .py extension to the Python file so that you can run it.

Then, add the following code to the file, save the file, and then run the file in the VS Code terminal.

counties = ["Arapahoe","Denver","Jefferson"]

if counties[1] == 'Denver':

print(counties[1])

This code might appear straightforward, but let's break down what's happening here.

* In the second line, we create our if statement and provide the condition, counties[1] == 'Denver'.
* The double equals sign, ==, is a comparison operator, or Boolean operator, which means "equal to."
* Since the second item in the counties list, counties[1], is "equal to" 'Denver', the condition is met. "Denver" is printed to the terminal screen.

**REWIND**

In Python, we can access the items in a list using an index.

## Comparison Operators

**Comparison operators,** or **Boolean operators,** are used to compare values, which will return True or False according to the condition. The following table highlights the different comparison or Boolean operators and their meanings.

|  |  |
| --- | --- |
| **Operator** | **Meaning** |
| > | greater than |
| >= | greater than or equal to |
| < | less than |
| <= | less than or equal to |
| == | equal to |
| != | not equal to |

**NOTE**

For more information, see the [documentation on comparison operators in Python (Links to an external site.)](https://docs.python.org/3.7/library/stdtypes.html#comparisons).

## If-Else Statements

The if-else statement is used when we need an outcome for both true and false conditions. The if-else statement is also referred to as a **dual-alternative decision statement**.

An if-else statement will execute one block of statements, or path, if a condition is true, or another block of statements if the condition is false. This can be illustrated with the following flowchart.

In Python, the general format for the if-else statement is to write a dual-alternative decision as follows:

if condition:

statement 1

statement 2

else:

statement 1

statement 2

Here's how an if-else statement works.

* The if-else statement begins with the word "if," followed by a condition.
* The condition is tested.
* If the condition is true, the block of indented statements following the if statement is executed, and the program moves out of the if-else statement.
* If the condition is false, the program skips to the else statement, the block of indented statements following the else statement is executed, and the program moves out of the if-else statement.

**IMPORTANT**

When writing an if-else statement, follow these general rules for indentation:

* Make sure the ifstatement and the else statement are aligned.
* Make sure the statements under the if and else statements are consistently indented.

To demonstrate how an if-else statement works, let's write a Python script that matches our flowcharts. Create a new Python file and add the following code. Then run the file in the VS Code terminal.

temperature = int(input("What is the temperature outside? "))

if temperature > 80:

print("Turn on the AC.")

else:

print("Open the windows.")

Here, the temperature variable is declared by asking the user to enter the outside temperature with an input statement wrapped in the int() method. The int() method will convert the user input data type from a string to an integer. The integer is used to assess the if-else statement.

## Nested If-Else Statements

Sometimes, a decision structure can be more complex than a dual-alternative decision structure. For instance, it's not uncommon for decision structures to be nested inside another decision structure. An example of this is writing an algorithm to determine a letter grade based on a number. Take a look at the following chart.

The general rules for the nested if-else decision structure example above are the same as for a single dual if-else statement.

To demonstrate how a nested if-else statement works, let's write a Python script that matches the flowchart. Create a new Python file and add the following code. Then run the file in the VS Code terminal.

#What is the score?

score = int(input("What is your test score? "))

# Determine the grade.

if score >= 90:

print('Your grade is an A.')

else:

if score >= 80:

print('Your grade is a B.')

else:

if score >= 70:

print('Your grade is a C.')

else:

if score >= 60:

print('Your grade is a D.')

else:

print('Your grade is an F.')

These nested if-else statement is quite complex and has a special name: if-elif-else statements, or the if-elif-else statement.

With nested if-elif-else statements, we can create a compound statement, elif, in the following manner.

* The first if statement tests the condition.
* The following else statement and the preceding if statement are compounded to make the elifstatement.
* This syntax continues until the last else statement.
* In an if-elif-else statement, the if, elif, and else statements are all aligned, and the conditionally executed blocks are indented.

Using these rules for the if-elif-else statement, let's rewrite the code and run the file in the VS Code terminal to determine a letter grade.

# What is the score?

score = int(input("What is your test score? "))

# Determine the grade.

if score >= 90:

print('Your grade is an A.')

elif score >= 80:

print('Your grade is a B.')

elif score >= 70:

print('Your grade is a C.')

elif score >= 60:

print('Your grade is a D.')

else:

print('Your grade is an F.')

Compared to the nested if-elif statement, the if-elif-else statement tends to be easier to read. This is because the if, elif, and else statements are aligned, and the block statements are usually shorter.

**HINT**

[Previous](https://courses.bootcampspot.com/courses/1225/modules/items/497141)[Next](https://courses.bootcampspot.com/courses/1225/modules/items/497149)

© 2020 - 2022 Trilogy Education Services, a 2U, Inc. brand. All Rights Reserved.

[Bootcamp: UCD-VIRT-DATA-PT-03-2022-U-B-MW](https://courses.bootcampspot.com/courses/1225)

[3.2.9: Membership and Logical Operators](https://courses.bootcampspot.com/courses/1225)

Immersive Reader

**3.2.9**

# Membership and Logical Operators

**As** Tom has demonstrated, decision statements play an important role in data retrieval. However, they can become more powerful and complex when you need to create compound comparison expressions or conditional statements. Situations may arise in which you may have to test two or more conditions in a data structure. This is where membership and logical operators come in handy. Next, Tom will show you how to test conditions using logical and membership operators.

## Membership Operators

**Membership operators** are used to test if an object, like a string, integer, or other data type is present in an expression, list, tuple, or dictionary. The following table describes these operators and provides an example of each one.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Meaning** | **Example** |
| in | Returns True if a sequence with the specified value is present in the object. | counties = ["Arapahoe","Denver","Jefferson"] if "Arapahoe" in counties: print("True") else: print("False")  This prints "True" because Arapahoe is in the counties list. |
| not in | Returns True if a sequence with the specified value is**not**present in the object. | counties = ["Arapahoe","Denver","Jefferson"] if "El Paso" not in counties: print("True") else: print("False")  This prints "True" because El Paso is not in the counties list. |

Let's practice creating a membership operation by determining if "El Paso" is in the counties list.

In the Python\_practice.py file, add the following code to your file and run the file.

counties = ["Arapahoe","Denver","Jefferson"]

if "El Paso" in counties:

print("El Paso is in the list of counties.")

else:

print("El Paso is not the list of counties.")

The output of this code will be:

El Paso is not the list of counties.

In the code, the decision statement checks if "El Paso" is in the counties list. If it is true, then the first print statement is printed to the screen. But the if statement is false, so the else statement is checked; since it is true, the second print statement is printed to the screen.

## Logical Operators

**Logical operators** allow us to connect multiple comparison expressions to create a compound expression. There are three logical operators: **and, or,** and **not.** The following table describes these operators and provides an example of each one.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Meaning** | **Example** |
| and | Evaluates two Boolean expressions into one compound expression. The compound expression is true if **both** Boolean expressions are true.  If one of the expressions is false, then the compound expression is false. | x = 5 y = 5 if x == 5 and y == 5: print("True") else: print("False")  This prints "True" because x = 5 is true and y = 5 is true. |
| or | Evaluates two Boolean expressions into one compound expression. The compound expression is true if **either** Boolean expression is true.  If one of the expressions is false, then the compound expression is true. If both expressions are false, then the compound expression is false. | x = 5 y = 5 if x == 3 or y == 5: print("True") else: print("False")  This prints "True" because x = 3 is false and y = 5 is true. |
| not | Evaluates a Boolean expression. The expression is true if the conditional is **false**. | x = 5 y = 5 if not(x > y): print("True") else: print("False")  This prints "True" because x is not greater than y. If x = 6, then it would print "False" because x is greater than y. |

We can combine membership and logical operations to test certain conditions. Let's practice creating a compound membership and logical operation using the list of counties.

We will use the "and" operator to determine if two counties, Arapahoe and El Paso, are in the list of counties.

if "Arapahoe" in counties and "El Paso" in counties:

print("Arapahoe and El Paso are in the list of counties.")

else:

print("Arapahoe or El Paso is not in the list of counties.")

The output of this code will be:

Arapahoe or El Paso is not in the list of counties.

In the code, the decision statement checks if both Arapahoe and El Paso are in the counties list. Arapahoe is in the counties list, which is "true," but El Paso is not in the counties list, which is "false." Therefore, the compound expression is false.

Next, let's check if either "Arapahoe" or "El Paso" is in the counties list. In the previous code, replace the logical operator "and" with "or" and run the file.

if "Arapahoe" in counties or "El Paso" in counties:

print("Arapahoe or El Paso is in the list of counties.")

else:

print("Arapahoe and El Paso are not in the list of counties.")

The output of this code will be:

Arapahoe or El Paso is in the list of counties.

The decision statement checks if either Arapahoe or El Paso is "true." We know that Arapahoe is in the counties list, which is "true," but El Paso is not in the counties list, which is "false." Therefore, the compound expression is "true," and the first print statement is executed.

If Arapahoe and El Paso are not in the list, then both expressions are "false." Therefore, the compound expression is "false," and the following statement is printed to the terminal.

Arapahoe and El Paso are not in the list of counties.

[Previous](https://courses.bootcampspot.com/courses/1225/modules/items/497146)[Next](https://courses.bootcampspot.com/courses/1225/modules/items/497155)

© 2020 - 2022 Trilogy Education Services, a 2U, Inc. brand. All Rights Reserved.

[Bootcamp: UCD-VIRT-DATA-PT-03-2022-U-B-MW](https://courses.bootcampspot.com/courses/1225)

[3.2.10: Repetition Statements](https://courses.bootcampspot.com/courses/1225)

Immersive Reader

**3.2.10**

# Repetition Statements

**Seth** asks if you and Tom have covered repetition statements yet. After looking at the dataset more closely, Seth thinks you'll be using repetition statements to retrieve data to complete the audit. Tom agrees and offers to walk you through how to create and use repetition statements.

When writing an algorithm that performs the same task over and over, it's a lot easier to write the code for the algorithm once, and then place that algorithm in a repetition structure that can be repeated as many times as necessary. This type of repetition structure is called a **loop**.

**REWIND**

Loops tell a computer to repeat lines of code over and over (and over) again and for loops tell the computer to repeat the lines of code a specific number of times.

There are two categories of repetition structures: condition-controlled loops and count-controlled loops.

A **condition-controlled loop** uses a true or false condition to control the number of times that it repeats, like asking a user if they want to continue with their online shopping after they add something to their "basket." This type of loop is also called a while loop.

A **count-controlled loop** repeats a specific number of times depending on the conditions, such as getting all the items in a list. This type of loop is also called a for loop.

Let's go over each loop type in more detail.

## While Loops

while loops test if a condition is true. If the condition is true, then the code will perform a task. This type of loop has two parts:

1. A condition that is tested for a true or false value.
2. A statement or statements that are repeated as long as the condition is true.

See the following flowchart that illustrates this process.

Let's practice creating and implementing a while loop. Create a new Python file in VS Code and add the following code to your file.

x = 0

while x <= 5:

print(x)

x = x + 1

Let's take a closer look at what this code is doing.

* We set the variable x = 0 before we enter the loop.
* We test if x is less than or equal to 5.
* If this condition is true, then we print the value of x.
* With x = x + 1, we increment x by 1.
* The condition is tested again

We repeat this process until the condition is false. When x is greater than 5, the loop stops.

**CAUTION**

If you forget to write code inside the loop that makes the test condition false, the while loop will continue to run. This is called an **infinite loop.** An infinite loop continues to repeat until the program is interrupted, either by quitting tor exiting the program. Avoid writing infinite loops!

## For Loops

for loops iterate, or run through, a program a specific number of times before it stops. A for loop can be written like if and if-else statements. Here's the general format:

for item in [items]:

statement 1

statement 2

Let's break this down step by step.

* There is a for statement, followed by an item or variable that is found in a list of items of an unknown number. The items can be a list, as in the previous code block example, or a tuple or dictionary.
* Upon the first iteration, the item will be the first item in a list. The second iteration will be the second item in the list, and so on.
* The statements that follow will be executed as long as the number of items has not been exhausted. When there are no more items in the list, the for loop stops.

### Iterate Through Lists and Tuples

Let's practice creating and implementing a for loop by iterating through our list of counties. Add the following code to the Python\_practice.py file.

for county in counties:

print(county)

When we execute this code, the county variable is declared and set equal to the first item in the list of counties, "Arapahoe." Then we print the first item, "Arapahoe," to the screen. For the second iteration, the county variable is set equal to "Denver," and then "Denver" is printed. This process continues for the number of items in the list of counties.

Arapahoe

Denver

Jefferson

Python has a built-in function, range(), that simplifies the process of writing a for loop. The range() function creates an iterable object or a list. For example, if we had a list of numbers, we could print each number using a for loop like this:

numbers = [0, 1, 2, 3, 4]

for num in numbers:

print(num)

When this code is executed, all the numbers in the list would be printed to the screen.

0

1

2

3

4

We will get the same output if we simplify the code by modifying the for loop, using the range() function:

for num in range(5):

print(num)

Indexing can also be used to iterate through a list. If the list contains strings, we'll need to get the length of the list as an integer for the range() function. For example, to iterate through the counties list using the range() function, the code should be rewritten as follows:

for i in range(len(counties)):

print(counties[i])

Let's break down what's happening in this code.

* The variable i is used to indicate the index, or the values 0, 1, and 2, in the length of the counties list. The letter i is often used for simplicity, but any variable can be used.
* Inside the range() function, we get the length of the list of counties, which is the integer 3.
* Then, we iterate through the list, where the variable i is equal to 0 for the first index. The 0 is passed into the print(counties[i]) statement, where i = 0, and "Arapahoe" is printed.
* This process is continued until all three items, or counties, in the list are printed to the screen.

**NOTE**

A variable being used to iterate through a for loop is chosen arbitrarily and could be anything, but it's a best practice to use a variable that makes sense in the current situation.

We can iterate through a tuple the same way we iterate through a list.

### Iterate Through a Dictionary

We can use a for loop to iterate over a dictionary and get all the keys, all the values, or all the keys and values.

To practice getting the keys and values from a dictionary, we will use the counties dictionary that was created earlier:

counties\_dict = {"Arapahoe": 422829, "Denver": 463353, "Jefferson": 432438}

### Get the Keys of a Dictionary

To get only the keys from a dictionary using a for loop, the loop can be written like we are iterating over a list or tuple.

for county in counties\_dict:

print(county)

When we execute this code, we get the counties printed to the screen.

Arapahoe

Denver

Jefferson

We can also use the keys() method to iterate over a dictionary to get the keys. To the previous code, add the keys() method to the end of the counties\_dict, like this:

for county in counties\_dict.keys():

print(county)

When using the keys() method, it doesn't matter what variable name we use in the for loop. The keys() method will print each key in order.

**SKILL DRILL**

Print each county from the counties dictionary using the keys() method.

End of text box.

### Get the Values of a Dictionary

To get the values of a dictionary, we iterate over the dictionary using the values() method, just like we used the keys() method.

for voters in counties\_dict.values():

print(voters)

Also, when using the values() method, it doesn't matter what variable name we use in the for loop. The values() method will print each value in order.

**REWIND**

You can also use the format dictionary\_name[key] to get the value for the key.

When using the format dictionary\_name[key], include the key county in the for loop in the print statement. This will return the value of the key in the output.

For the counties list, modify the for loop and use the key, county to reference the value.

for county in counties\_dict:

print(counties\_dict[county])

Another method we can use to get the values of a key is to use the get() method on the dictionary in the for loop.

for county in counties\_dict:

print(counties\_dict.get(county))

The output of both of these for loops will be:

422829

463353

432438

### Get the Key-Value Pairs of a Dictionary

Finally, if we want to print the key-value pair of the dictionary, we use the items() method in the for loop, which follows this general format:

for key, value in dictionary\_name.items():

print(key, value)

When we use the items() method, we get the key and the value for each dictionary by referencing them as "key" and "value" as in the code above, or we can reference them by name, like "county" and "voters", as in the code below.

for county, voters in counties\_dict.items():

print(county, voters)

When we execute this code in the VS Code terminal, the output will be each key and value in order.

Arapahoe 422829

Denver 463353

Jefferson 432438

**IMPORTANT**

When iterating over a dictionary:

* The first variable declared in the for loop is assigned to the keys.
* The second variable is assigned to the values.

**SKILL DRILL**

Print each county and registered voter form the counties dictionary so that the output looks like this:

End of text box.

### Iterate Through a List of Dictionaries

A for loop can be used to iterate through a list of dictionaries like our voting\_data list of dictionaries we created earlier. With a for loop we can:

* Retrieve each dictionary in the list
* Retrieve only the values of each dictionary
* Retrieve the key-value pairs of each dictionary

### Get Each Dictionary in a List of Dictionaries

To print each dictionary in voting\_data, use the standard format for iterating over the list of dictionaries with a for loop:

voting\_data = [{"county":"Arapahoe", "registered\_voters": 422829},

{"county":"Denver", "registered\_voters": 463353},

{"county":"Jefferson", "registered\_voters": 432438}]

for county\_dict in voting\_data:

print(county\_dict)

When we execute the code, each dictionary is printed on a separate line:

{'county': 'Arapahoe', 'registered\_voters': 422829}

{'county': 'Denver', 'registered\_voters': 463353}

{'county': 'Jefferson', 'registered\_voters': 432438}

Since this is a list of dictionaries, use the range() function to iterate over the list.

### Get the Values from a List of Dictionaries

To retrieve only the values from each dictionary in the list of dictionaries, we need to use a **nested** for loop. Let's see how this can be done with our voting\_data.

First, use the for loop: for county\_dict in voting\_data: to retrieve each dictionary. Then, in the second for loop, use the values() method on the variable county\_dict to reference the data in the second for loop in order to get each value.

for county\_dict in voting\_data:

for value in county\_dict.values():

print(value)

When we execute this code in the VS Code terminal, the output is each value from each key:

Arapahoe

422829

Denver

463353

Jefferson

432438

If we only want to print the county name from each dictionary, we can use county\_dict['county'] in the print statement for the for loop.

for county\_dict in voting\_data:

print(county\_dict['county'])

When this code is executed, we retrieve each county in each dictionary.

Arapahoe

Denver

Jefferson

[Previous](https://courses.bootcampspot.com/courses/1225/modules/items/497149)[Next](https://courses.bootcampspot.com/courses/1225/modules/items/497159)

© 2020 - 2022 Trilogy Education Services, a 2U, Inc. brand. All Rights Reserved.

[Bootcamp: UCD-VIRT-DATA-PT-03-2022-U-B-MW](https://courses.bootcampspot.com/courses/1225)

[3.2.11: Printing Formats](https://courses.bootcampspot.com/courses/1225)

Immersive Reader

**3.2.11**

# Printing Formats

**While** performing the election analysis, you and Tom will have to test your code with Python print statements. Learning how to use a variety of printing formats and adding print statements in your Python scripts is a common practice among programmers. Creating print statements will you help determine if your code is correct when you run it.

As you have already seen in your work so far, there are several ways to print information to the output screen. Knowing how to print in a variety of formats and how to format floating-point decimal numbers is useful for programmers and is often used when writing any Python script.

The different print methods we will cover are the print() function, printing single and multiline f-strings, and formatting numbers in print statements.

## The print() Function

So far, we have used the Python print() function to print the following:

1. A string, or a sentence displayed between quotes. For example: print("Hello World") and print("Arapahoe and Denver are not in the list of counties.").
2. A string with integer values converted to a string using concatenation with the "+" sign. For example: print("Your interest for the year is $" + str(interest)) .

Both of these are sufficient when printing simple statements, but using concatenation can become cumbersome when we need to print items from a list of values from a dictionary. This is where f-strings come in.

## F-strings: Formatted String Literals

With Python 3.7, printing has become much easier with the use of **f-string literals,** which can be used in place of concatenation. The general format for f-strings is as follows:

1. The f-string begins with an f followed by a string contained within quotes. (The term f-string comes from the leading "f" character preceding the string literal.)
2. In the f-string, curly braces are used to add variables or expressions to the f-string.

To see an example, let's edit the code we wrote to calculate the percentage of votes using f-string literals.

Here's the original code:

my\_votes = int(input("How many votes did you get in the election? "))

total\_votes = int(input("What is the total votes in the election? "))

percentage\_votes = (my\_votes / total\_votes) \* 100

print("I received " + str(percentage\_votes)+"% of the total votes.")

And here's how you would edit the code to use f-strings.

my\_votes = int(input("How many votes did you get in the election? "))

total\_votes = int(input("What is the total votes in the election? "))

print(f"I received {my\_votes / total\_votes \* 100}% of the total votes.")

Look at the last line in particular. Inside the curly braces, the f-string performs the calculation my\_votes / total\_votes \* 100 and formats the value as a string. There is no need to convert percentage\_votes in the original code to the string format. This makes the code more concise and easier to read!

## Using F-Strings with Dictionaries

F-strings can be used to print out the keys or values of a dictionary. This will make our code easier to write and read.

Let's edit the code you may have written for the Skill Drill where you needed to print each county and registered voter from the counties dictionary.

Here's the counties dictionary and the solution for that Skill Drill if we use concatenation.

counties\_dict = {"Arapahoe": 369237, "Denver":413229, "Jefferson": 390222}

for county, voters in counties\_dict.items():

print(county + " county has " + str(voters) + " registered voters.")

The output should when this code is executed should look like this:

Arapahoe county has 369237 registered voters.

Denver county has 413229 registered voters.

Jefferson county has 390222 registered voters.

If we use f-stings, we can rewrite the print statement to be more intuitive and clear.

for county, voters in counties\_dict.items():

print(f"{county} county has {voters} registered voters.")

## Multiline F-Strings

Another use for f-strings is to print multiple strings or lines to the screen. Let's say you need to tell a candidate how many votes they won, the total number of votes, and the percentage of votes they received. You can use the code you wrote to calculate the percentage of votes and create a message to be printed to a screen, like this:

candidate\_votes = int(input("How many votes did the candidate get in the election? "))

total\_votes = int(input("What is the total number of votes in the election? "))

message\_to\_candidate = (

f"You received {candidate\_votes} number of votes. "

f"The total number of votes in the election was {total\_votes}. "

f"You received {candidate\_votes / total\_votes \* 100}% of the total votes.")

print(message\_to\_candidate)

When we run this file in VS Code, the output will look like this when you input 3,345 for the candidate's votes and 23,123 for the total votes.

You received 3345 number of votes. The total number of votes in the election was 23123. You received 14.466115988409808% of the total votes.

## Format Floating-Point Decimals

Notice that in the output from the algorithm above, the percentage of votes is, 14.466115988409808! In Python, we can format numbers with a thousands separator and specify a decimal precision.

The general format for a number to format it in an f-string is as follows:

f'{value:{width}.{precision}}'

In the format, the width specifies the number of characters used to display the value. However, if a value needs more space than the width specifies, the additional space is used.

The precision indicates the number of decimal places to format the value. For example, to format the interest to two decimal places, we would use .2f, where f means "floating-point decimal format".

When formatting a number, we can also add a thousands separator with a comma, using the following format. The comma is placed after the {width}.

f'{value:{width},.{precision}}'

Let's add a thousands separator to the output for the candidate votes and total votes and then format the percentage of votes to two decimal places. The code should look like this:

message\_to\_candidate = (

f"You received {candidate\_votes:,} number of votes. "

f"The total number of votes in the election was {total\_votes:,}. "

f"You received {candidate\_votes / total\_votes \* 100:.2f}% of the total votes.")

When this file is run in VS Code, the output will look like this when you input 3,345 for the candidate's votes and 23,123 for the total votes.

You received 3,345 number of votes. The total number of votes in the election was 23,123. You received 14.47% of the total votes.

**SKILL DRILL**

Refer to the following dictionary to complete the activity.

counties\_dict = {"Arapahoe": 422829, "Denver": 463353, "Jefferson": 432438}

Print each county and registered voter from the dictionary. The output should look like the following:

End of text box.

**SKILL DRILL**

Refer to the following dictionary to complete the activity.

voting\_data = [``{"county":"Arapahoe", "registered\_voters": 422829}, {"county":"Denver", "registered\_voters": 463353}, {"county":"Jefferson", "registered\_voters": 432438}]

Print each county and registered voter from the dictionary. The output should look like the following:

End of text box.

**HINT**

**NOTE**

For more information, see the [documentation on formatting f-string literals (Links to an external site.)](https://www.python.org/dev/peps/pep-0498/).

Congratulations, you made it through this programming-packed day! That was a lot of information thrown at you in a short time. We recommend more practice so that you can keep these concepts fresh in your memory.

[Previous](https://courses.bootcampspot.com/courses/1225/modules/items/497155)[Next](https://courses.bootcampspot.com/courses/1225/modules/items/497168)

© 2020 - 2022 Trilogy Education Services, a 2U, Inc. brand. All Rights Reserved.

[Bootcamp: UCD-VIRT-DATA-PT-03-2022-U-B-MW](https://courses.bootcampspot.com/courses/1225)

[3.3.1: Import and Inspect the Data](https://courses.bootcampspot.com/courses/1225)

Immersive Reader

**3.3.1**

# Import and Inspect the Data

**Before** stopping for the day, you take a look at the dataset again with Tom. You want to be familiar with the data before you write any code. Seth tells you to download the dataset and save it in your Election\_Analysis folder, and then go see Tom to inspect the data.

## Import the Data

The first step is to import the data. In your Election\_Analysis folder, create a subfolder named "Resources." This is where you'll save the dataset.

Click the following link to download the dataset into the Resources folder.

[Download the dataset (Links to an external site.)](https://2u-data-curriculum-team.s3.amazonaws.com/dataviz-online/module_3/election_results.csv)

When the download is complete, you should see a file named election\_results.csv in the Resources folder.

## Inspect the Data

A **CSV file** is one of the most common data formats that is used with large datasets. When data is exported from websites or databases, they are primarily exported in the form of a CSV file.

CSV stands for comma-separated value. This means that each value is separated by a comma. If you open election\_results.csv with a text editor like VS Code, you'll see that each row, or value, is separated by a comma.

Before we do any data analysis, we need to inspect the data.

**REWIND**

Remember, when inspecting the data focus on the following questions.

* How many columns and rows are there?
* What types of data are present?
* Is the data readable, or does it need to be converted in some way?

Open election\_results.csv in Excel so you can see the number of columns and rows. A text editor will not give this level of detail. You may notice that this file looks very similar to an Excel file with an .xlsx extension. You should also notice that there are three headers: Ballot ID, County, and Candidate. Here are the first 10 rows:

**NOTE**

Sometimes CSV files can have **headers**. Headers are essentially a way to label each column of data in the file.

If we scroll down to the last row, you'll see there are 369,712 rows.

**REWIND**

There's a keyboard shortcut to get to the last row of an Excel file so that you don't have to keep scrolling. Place the cursor in a column that doesn't contain any empty cells and press Command and the down arrow key (for macOS) or CTRL and the down arrow key (for Windows).

The data consists of a number for the ballot ID and a name for the county and candidate, respectively. At this point, we don't know how many counties and candidates are in the file. Overall, the data appears readable and there are no unusual row values that we can see—but let's use Python to check anyway.

[Previous](https://courses.bootcampspot.com/courses/1225/modules/items/497159)[Next](https://courses.bootcampspot.com/courses/1225/modules/items/497172)

© 2020 - 2022 Trilogy Education Services, a 2U, Inc. brand. All Rights Reserved.

[Bootcamp: UCD-VIRT-DATA-PT-03-2022-U-B-MW](https://courses.bootcampspot.com/courses/1225)

[3.3.2: Overview of the Project](https://courses.bootcampspot.com/courses/1225)

Immersive Reader

**3.3.2**

**Overview of the Project**

**Now** that you have seen the dataset, you're going to sit down with Tom to go over the tasks that need to be completed for the election audit, and discuss the information needed by the Colorado Board of Elections.

In this project, our final Python script will need to be able to deliver the following information when the script is run:

* Total number of votes cast
* A complete list of candidates who received votes
* Total number of votes each candidate received
* Percentage of votes each candidate won
* The winner of the election based on popular vote

[Previous](https://courses.bootcampspot.com/courses/1225/modules/items/497168)[Next](https://courses.bootcampspot.com/courses/1225/modules/items/497177)

© 2020 - 2022 Trilogy Education Services, a 2U, Inc. brand. All Rights Reserved.

[Bootcamp: UCD-VIRT-DATA-PT-03-2022-U-B-MW](https://courses.bootcampspot.com/courses/1225)

[3.3.3: Pseudocoding](https://courses.bootcampspot.com/courses/1225)

Immersive Reader

**3.3.3**

**Pseudocoding**

**After** giving you an overview of the election audit tasks, Tom wants to go over the steps required in detail. He's going to show you a technique commonly used by programmers to write steps of their code, which is called **pseudocode**. Pseudocode will make the audit easier to present to nontechnical colleagues and stakeholders.

To facilitate the design process, programmers use **pseudocode** to create models or flowcharts for their programs. Pseudocode is like a roadmap of what you think your code will look like or the steps you'll take to complete the task at hand.

*Pseudo* means "fake," so pseudocode is essentially fake code. Pseudocode is an informal language that has no syntax rules and is not meant to be executed. The point of using pseudocode is to focus on the overall design of the program.

For example, let's say someone asks you how to wash clothes. You might break down this task into a series of basic steps, like this:

1. Open the lid of the washing machine.
2. Put clothes in the washing machine.
3. Turn on the water.
4. Add detergent.
5. Close the lid.

These well-defined, logical steps that are sequentially ordered is an example of an **algorithm**.

Similarly, a programmer or analyst may write the steps to count the votes of an election like this:

1. Open the data file.
2. Write down the names of all the candidates.
3. Add a vote count for each candidate.
4. Get the total votes for each candidate.
5. Get the total votes cast for the election.

A good place to start with when writing pseudocode is to think about the end goal. Then, consider the steps you need to take in order to reach that end goal. It's fine if you don't know all of the steps at first; this will come with practice. Also, the more you write pseudocode, the more of a feel you'll get for planning and design using pseudocode. Simply put, there is no right or wrong way to write pseudocode, as long as you're breaking down the problem into smaller, more manageable problems to solve.

Let's practice writing pseudocode. Launch VS Code and create a new file named PyPoll.py. Create a high-level list of the necessary steps given to you using pseudocode.

Notice that in this pseudocode there is a hashtag, or number sign, before each comment: #. When the Python interpreter sees the # character, it ignores everything from that character to the end of the line. Using this technique is a great way to self-document your code so that others know what you are doing. We will have more practice at documenting our code later in this module.

[Previous](https://courses.bootcampspot.com/courses/1225/modules/items/497172)[Next](https://courses.bootcampspot.com/courses/1225/modules/items/497186)

© 2020 - 2022 Trilogy Education Services, a 2U, Inc. brand. All Rights Reserved.

[Bootcamp: UCD-VIRT-DATA-PT-03-2022-U-B-MW](https://courses.bootcampspot.com/courses/1225)

[3.4.1: Python Dependencies, Modules, and Packages](https://courses.bootcampspot.com/courses/1225)

Immersive Reader

**3.4.1**

# Python Dependencies, Modules, and Packages

**After** inspecting the dataset using Excel, you tell Tom and Seth that you are ready to open the file and begin the election audit. Seth explains to you that before you open the CSV, you need to make a connection to the file by using your computer's directory "path" to that file. Once you connect to your file, you will be able to access the contents of the file. However, you need to use a programming tool for this specific purpose. Tom will show you how to connect to the file and then walk you through how to use some programming tools that will give you access to the contents in the CSV file.

For data analysis, the process of opening and reading files is important. All programming languages have a set of methods that will open a file and read the file. The programming language you are using requires very precise directions for the path to the file.

The election\_results.csv file should be located in the Resources folder, as shown in the following image.

If we were writing code in PyPoll.py, the path to the CSV can be written Resources/election\_results.csv. If  election\_results.csv was not in the Resources folder but at the same "level" as PyPoll.py, then the path would be election\_results.csv.

**IMPORTANT**

Different operating systems use different path separators to separate files and folders.

macOS: Use a forward slash to separate folders and files. "/"

Windows: Uses backslashes to separate folders and files. "\"

Now that we know how to connect to our election\_results.csv file, we need to read, process, and parse, or analyze, the data in the file. To do this will require us to use programming tools like you would use hardware tools in a toolbox. These tools, in programming parlance, are called dependencies, packages, and modules.

## Dependencies

**Dependencies** are modules and packages, or a programming script that someone else has written, that allows you to increase the functional programming of your code, or speed and efficiency.

Python is an open-source language, which means that it is easy for others to write dependencies that can be used with Python. You can think of the relationship between dependencies, modules, and packages like Russian nesting dolls.

Dependencies are the largest "doll," like the Python **datetime module**. Inside the datetime module are functions, classes, or variables, which are the second-largest doll. The methods used for functions and classes are in the third-largest doll.

This is how we would use the datetime module to get today's date using VS Code.

# Import the datetime class from the datetime module.

import datetime

# Use the now() attribute on the datetime class to get the present time.

now = datetime.datetime.now()

# Print the present time.

print("The time right now is ", now)

Let's break down what is happening in the code above as it relates to the dependency "datetime". The datetime module comes with our Python installation.

1. To use the datetime module all we need to do is to import it using import datetime.
2. In line 4, we declare the now variable to hold the time right "now".
3. The now variable is set equal to datetime.datetime.now(), where:
   * The first datetime is the datetime module, (first doll).
   * The second datetime is the datetime class (second doll).
   * Then we use the datetime attribute, now(), (third doll) on the datetime class, i.e., datetime.now(), to get the current time.

When we run this code, the output will be the current time at the moment the code is run, and will look similar to the following:

The time right now is 2019-09-18 14:11:42.394131

To reduce the confusion on importing a module with the same name as a class we can use an abbreviated alias dt for the datetime module.

# Import the datetime class from the datetime module.

import datetime as dt

# Use the now() attribute on the datetime class to get the present time.

now = dt.datetime.now()

# Print the present time.

print("The time right now is ", now)

## Packages

**Packages** are folders that contain a set of Python modules. The folders in the packages may contain various subpackages, or other folders. To import packages, we use the import statement, as we did with the datetime module.

## Modules

**Modules** are a separate software component. They are usually Python files with a .py extension. The name of the module will be the name of the file. A Python module can have functions, classes, or variables defined and implemented.

Modules can be used in a variety of applications and functions with other programs. They may contain hundreds or thousands of lines of code, so it would be foolish to write or repurpose the code every time you need to use it. This type of programming can lead to many syntax or logical errors in the program that would require an enormous amount of time to correct.

Modules are easy to use and maintain, and they provide reusability with a simple statement like import datetime. To use a specific function, class, or variable from a module, you use a statement like from import.

If your script requires the use of programs, modules, and packages, one of the first steps is to import dependencies for your Python script.

Let's put this concept to good use and walk through how to read a CSV file by using the CSV module.

## The CSV Module

In Python there's a built-in module called csv, which allows users to easily pull in data from external CSV files and perform operations on them.

The csv module is imported by using the import statement followed by the module name, csv.

The csv module has many functions that allow us to read and write tabular data in CSV format. With the csv module, we can read data from a file that was generated by Excel and write data to a file in a format that can be read by Excel.

To see all the functions available in the csv module, follow these steps:

1. Launch the Python interpreter.
2. Type import csv to import the module.
3. Press Enter.
4. Type dir(csv). The "dir" is short for "directory".

The Python interpreter should look like this:

>>> import csv

>>> dir(csv)

Press Enter. The output will look like this:

['Dialect', 'DictReader', 'DictWriter', 'Error', 'OrderedDict', 'QUOTE\_ALL', 'QUOTE\_MINIMAL', 'QUOTE\_NONE', 'QUOTE\_NONNUMERIC', 'Sniffer', 'StringIO', '\_Dialect', '\_\_all\_\_', '\_\_builtins\_\_', '\_\_cached\_\_', '\_\_doc\_\_', '\_\_file\_\_', '\_\_loader\_\_', '\_\_name\_\_', '\_\_package\_\_', '\_\_spec\_\_', '\_\_version\_\_', 'excel', 'excel\_tab', 'field\_size\_limit', 'get\_dialect', 'list\_dialects', 're', 'reader', 'register\_dialect', 'unix\_dialect', 'unregister\_dialect', 'writer']

If you look closely at the output, you'll see a function called reader. We'll use this function to read the CSV file that contains the election data.

Using the dir() function, we can pass:

1. A Python module, like the csv module. The dir() function will return all the functions available in the csv module.
2. A variable, like a dictionary {'key':'value'}, for example the counties\_dict dictionary. The dir() function will return all the functions available on that variable.
3. >>> dir({'Arapahoe': 422829, 'Denver': 463353, 'Jefferson': 432438})

['\_\_class\_\_', '\_\_contains\_\_', '\_\_delattr\_\_', '\_\_delitem\_\_', '\_\_dir\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattribute\_\_', '\_\_getitem\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_init\_\_', '\_\_init\_subclass\_\_', '\_\_iter\_\_', '\_\_le\_\_', '\_\_len\_\_', '\_\_lt\_\_', '\_\_ne\_\_', '\_\_new\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_setattr\_\_', '\_\_setitem\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_subclasshook\_\_', 'clear', 'copy', 'fromkeys', 'get', 'items', 'keys', 'pop', 'popitem', 'setdefault', 'update', 'values']

1. A data type, like str. The dir() function will return all the attributes and methods that can be used with the str data type.

>>> dir(str) '\_\_add\_\_', '\_\_class\_\_', '\_\_contains\_\_', '\_\_delattr\_\_', '\_\_dir\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattribute\_\_', '\_\_getitem\_\_', '\_\_getnewargs\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_init\_\_', '\_\_init\_subclass\_\_', '\_\_iter\_\_', '\_\_le\_\_', '\_\_len\_\_', '\_\_lt\_\_', '\_\_mod\_\_', '\_\_mul\_\_', '\_\_ne\_\_', '\_\_new\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_rmod\_\_', '\_\_rmul\_\_', '\_\_setattr\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_subclasshook\_\_', 'capitalize', 'casefold', 'center', 'count', 'encode', 'endswith', 'expandtabs', 'find', 'format', 'format\_map', 'index', 'isalnum', 'isalpha', 'isdecimal', 'isdigit', 'isidentifier', 'islower', 'isnumeric', 'isprintable', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'maketrans', 'partition', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']

**SKILL DRILL**

For the following data types and data structures, use the dir() function to find all the attributes and methods.

1. int
2. float
3. bool
4. list
5. tuple
6. dict
7. datetime

Here are some other modules that may be useful later. You will have to import these first.

* random
* numpy

End of text box.

[Previous](https://courses.bootcampspot.com/courses/1225/modules/items/497177)[Next](https://courses.bootcampspot.com/courses/1225/modules/items/497189)

© 2020 - 2022 Trilogy Education Services, a 2U, Inc. brand. All Rights Reserved.

[Bootcamp: UCD-VIRT-DATA-PT-03-2022-U-B-MW](https://courses.bootcampspot.com/courses/1225)

[3.4.2: Open and Read Files Using Python](https://courses.bootcampspot.com/courses/1225)

Immersive Reader

**3.4.2**

# Open and Read Files Using Python

**Now** that you know how to bring in the programming tools, or import dependencies, you're ready to practice opening and reading CSV files. There are two ways to read a file in using a programming language: supply a direct path to the file or use an indirect path. Being familiar with both methods will help you avoid setbacks later on during other projects, Tom will review both with you.

During the file handling process, i.e., opening and reading a file, we may come across two different file types: a text file and a binary file. Before we walk through how to open a file let's go over these two file types.

## File Types

A **text file** can be opened in a text editor like VS Code, TextEdit on Mac, and Notepad on Windows. A CSV file is a text file. The data in a text file is encoded as text using ASCII (pronounced "askee") or Unicode. A **binary file** contains data that has not been converted to text. Binary files cannot be opened with a text editor.

**REWIND**

Binary numbers, or base-2 numbers, are numbers that are written with only ones and zeros. In binary, we use the digits 0 and 1 to represent how many times a power of 2 is included in a number.

**NOTE**

For more information, go to the [ASCII website (Links to an external site.)](https://ascii.cl/) and this [website on Unicode characters (Links to an external site.)](https://www.rapidtables.com/code/text/unicode-characters.html).

## Open a File

You can access a file in a folder on your computer if you know the direct file path. If you do not know the direct file path, but know the folder and filename, you can access the file indirectly.

After providing the file path in our Python script, we will be able to open and read the file. When the program reads the file, it creates a **file object** in the computer's memory, which provides a way for the program to work with that file. In our script, we can use a variable to reference the file object.

The general format for opening a file is, file\_variable = open(filename, mode).

Let's break down what each component is doing in the general format.

* file\_variable is the name of the variable that will reference the file object.
* filename is a string specifying the name of the file.
* mode is a string specifying the mode for reading or writing the file object. The possible modes are:
  + "r": Open a file to be read.
  + "w": Open a file to write to it. This will overwrite an existing file and create a file if one does not already exist.
  + "x": Open a file for exclusive creation. If the file does not exist, it will not create one.
  + "a": Open a file to append data to an existing file. If a file does not exist, it creates one, if a file has been created the data will be added to the file.
  + "+": Open a file for reading and writing.

Now that we know how to open a file, we need to open our election\_results.csv file and read the data in the file.

## Read Data from a File

Let's go over how to read data from a file using both a direct path and an indirect path.

#### Direct Path to the File

First, open the CSV file using the direct path method. The direct path to our election\_results.csv file will be Resources/election\_results.csv.

Using VS Code, type the following in the PyPoll.py file to assign a variable to our file path.

# Assign a variable for the file to load and the path.

file\_to\_load = 'Resources/election\_results.csv'

When we type Resources/election\_results.csv, we are telling the computer to get the election\_results.csv file that is located in the "Resources" folder.

Next, we will open the file, file\_to\_load, with the open() function, using the "r" mode to read the file. Then, we'll print the filename object. After reading the file,  close the file with the close() function. In between the opening and closing of the file is where we will read the data and perform our analysis.

Below our file assignment variable, file\_to\_load, add the following code:

# Open the election results and read the file.

election\_data = open(file\_to\_load, 'r')

# To do: perform analysis.

# Close the file.

election\_data.close()

**IMPORTANT**

Closing a file disconnects the program from the file. It's important that you close the file after you read a file and write data to a file.

When you read data from a file and it is not closed at the end of the operation, you can lose some of the data. When you write data to a file, the data is not stored in the file at first. It is written to a "buffer" in the computer memory and may be overwritten later if the file is not closed. Once you close the file, the data is stored in the file.

Python has a way to read and write to a file without needing to use the open() and close() functions every time. We simply replace the open() function with the with statement.

The with statement opens the file and ensures proper acquisition or release of any data without having to close the file, to ensure that the data isn't lost or corrupted.

The format for the with statement is the following:

with open(filename) as file\_variable:

The file\_variable is used to reference the file object throughout the script.

Let's modify this code, using the with statement instead of the open() and close() functions. We'll print the file variable, election\_data, to the screen.

Edit your code below the file assignment variable, file\_to\_load, to look like this:

# Open the election results and read the file

with open(file\_to\_load) as election\_data:

# To do: perform analysis.

print(election\_data)

The with statement ends with a colon, which means we need to indent on the next line, as we did with if-else statements and for loops.

Save the PyPoll.py file and run the file in the VS Code terminal. The output in VS Code will look something like this:

<\_io.TextIOWrapper name='Resources/election\_results.csv' mode='r' encoding='UTF-8'>

In this output, the \_io.TextIOWrapper is a Python class that will allow us to read or write data to and from the file when we used the appropriate methods and attributes. The name represents the path of the file object, and the computer tells us that the file is open in "read" mode with UTF-8 encoding.

#### Indirect Path to the File

Sometimes we won't know the direct path to the file on our computer, only that it's in a specific folder. Usually, you will know the direct path, but in a real-world setting, you may be given the indirect path to the file by a fellow coworker or your manager.

To access and open a file for which the direct path is unknown, we use the os module.

The os module allows us to interact with our operating system. We can see all the different attributes and methods that the os module uses by importing the module and typing print(dir(os)) in the Python interpreter.

>>> import os

>>> dir(os)

The list is quite extensive, as you can see.

Python provides a submodule os.path that allows us to access files on different operating systems, like macOS and Windows.

The os.path submodule contains several useful functions to make it easier to join a path, as shown by typing dir(os.path) in the Python interpreter.

['\_\_all\_\_', '\_\_builtins\_\_', '\_\_cached\_\_', '\_\_doc\_\_', '\_\_file\_\_', '\_\_loader\_\_', '\_\_name\_\_', '\_\_package\_\_', '\_\_spec\_\_', '\_get\_sep', '\_joinrealpath', '\_varprog', '\_varprogb', 'abspath', 'altsep', 'basename', 'commonpath', 'commonprefix', 'curdir', 'defpath', 'devnull', 'dirname', 'exists', 'expanduser', 'expandvars', 'extsep', 'genericpath', 'getatime', 'getctime', 'getmtime', 'getsize', 'isabs', 'isdir', 'isfile', 'islink', 'ismount', 'join', 'lexists', 'normcase', 'normpath', 'os', 'pardir', 'pathsep', 'realpath', 'relpath', 'samefile', 'sameopenfile', 'samestat', 'sep', 'split', 'splitdrive', 'splitext', 'stat', 'supports\_unicode\_filenames', 'sys']

In this output, we can see there is a function called join. The join() function joins our file path components together when they are provided as separate strings; then, it returns a direct path with the appropriate operating system separator, forward slash for macOS or backward slash for Windows.

**REWIND**

Different operating systems use different path separators to separate files and folders:

* macOS uses the forward-slash: /
* Windows uses the backslash: \

To declare a variable for the file to load, connect the os.path submodule with the join() function, like this: os.path.join(). This is called chaining.

**Chaining** is a programmatic style that is used for making multiple method calls on the same object. This is a common practice that makes code look clean and concise.

Inside the parentheses of the join() function, we will add the folder and file to join together. In this case, we'll add the Resources folder and election\_results.csv separated by a comma, like this:

os.path.join("Resources", "election\_results.csv")

Then, we use a filename variable to reference the path to election\_data.csv, like this:

file\_to\_load = os.path.join("Resources", "election\_results.csv")

Let's put all of this to practical use! In the VS Code PyPoll.py file, complete the following steps:

1. Import the csv and os modules.
2. Add the filename variable that references the path to election\_results.csv.
3. Open the election\_results.csv using the with statement as the filename object, election\_data.
4. Print the filename object.

Your PyPoll.py file should look like this:

import csv

import os

# Assign a variable for the file to load and the path.

file\_to\_load = os.path.join("Resources", "election\_results.csv")

# Open the election results and read the file.

with open(file\_to\_load) as election\_data:

# Print the file object.

print(election\_data)

When we run this file in the VS Code terminal, the output is similar to when we used the direct path for the file variable:

<open file 'Resources/election\_results.csv', mode 'r' at 0x10479c780>

**IMPORTANT**

You'll notice that we made comments before the code to explain what we were doing. This is a good practice in coding. Not only will this help others reading your code, but it will also help you refresh your memory if you have to revisit your code a few months down the line.

**NOTE**

For more information, see the [documentation on file and directory access (Links to an external site.)](https://docs.python.org/3.7/library/os.path.html).

[Previous](https://courses.bootcampspot.com/courses/1225/modules/items/497186)[Next](https://courses.bootcampspot.com/courses/1225/modules/items/497195)

© 2020 - 2022 Trilogy Education Services, a 2U, Inc. brand. All Rights Reserved.

[Bootcamp: UCD-VIRT-DATA-PT-03-2022-U-B-MW](https://courses.bootcampspot.com/courses/1225)

[3.4.3: Write to Files with Python](https://courses.bootcampspot.com/courses/1225)

Immersive Reader

**3.4.3**

**Write to Files with Python**

**Seth** thinks this would be a good time for Tom to show you how to write data to a file. Writing data to a file requires similar steps as when we read a file, and is just as important: after you perform your analysis, you will need to write the results to a text file, which will be sent to the election commission.

To write a file to a directory on your computer, perform steps similar to those we followed when we read a file:

1. Create a filename variable to a direct or indirect path where the file is to be located.
2. Use the open() function in the "w" mode to open a file and write data to the file.

Here's how we would write this in Python.

# Create a filename variable to a direct or indirect path to the file.

file\_to\_save = os.path.join("analysis", "election\_analysis.txt")

# Using the open() function with the "w" mode we will write data to the file.

open(file\_to\_save, "w")

When you add this code to the PyPoll.py file and run it in VS Code terminal, we'll see that this results in an error.

Traceback (most recent call last):

File "PyPoll.py", line 24, in <module>

open(file\_to\_save, "w")

IOError: [Errno 2] No such file or directory: 'analysis/election\_analysis.txt'

The error is an IOError, which is an "Input/Output" error, meaning that we used an output directory, 'analysis/election\_analysis.txt', that doesn't exist with the given file path.

The error states that there is no file or directory 'analysis/election\_analysis.txt'. This error occurs because we don't have a folder named "analysis" where the election\_analysis.txt file should be saved.

To correct this error, create an empty folder in the Election\_Analysis folder and name it "analysis." When we execute the file again, we can open the "analysis" folder and see the election\_analysis.txt file in the VS Code sidebar.

Open election\_analysis.txt and you'll see that it's empty. As we perform the election analysis, we'll write data to this file. For now, let's practice adding some simple data to this file and saving it in the "analysis" folder.

In election\_analysis.txt add "Hello World" to the first line by adding the following code to PyPoll.py and running the file in VS Code.

# Create a filename variable to a direct or indirect path to the file.

file\_to\_save = os.path.join("analysis", "election\_analysis.txt")

# Use the open statement to open the file as a text file.

outfile = open(file\_to\_save, "w")

# Write some data to the file.

outfile.write("Hello World")

# Close the file

outfile.close()

Here's what's happening in this code:

1. After we create the file\_to\_save variable, we set the open(file\_to\_save, "w") to a filename variable, outfile.
2. Then, we use the filename variable to write "Hello World" to the file using the write() function from the os module.
3. Lastly, we use outfile.close() to close the file.

When we execute this file and open election\_analysis.txt, we see the string Hello World in the first line.

Now that we know how to write data to a file, let's make the code cleaner and more concise. We'll do this by using the with statement instead of using the open() and close() functions.

Your code for writing to a file should look like this:

# Create a filename variable to a direct or indirect path to the file.

file\_to\_save = os.path.join("analysis", "election\_analysis.txt")

# Using the with statement open the file as a text file.

with open(file\_to\_save, "w") as txt\_file:

# Write some data to the file.

txt\_file.write("Hello World")

We can continue to add data to this file using the write() function.

In place of "Hello World," add the following counties to the file: "Arapahoe," "Denver," and "Jefferson." This can be done in two ways.

The first way is to add each county with its own write() function on a separate line, like this:

# Write three counties to the file.

txt\_file.write("Arapahoe")

txt\_file.write("Denver")

txt\_file.write("Jefferson")

When we execute the file and open election\_analysis.txt we'll see the names of the three counties, but there is no space between each county name.

To separate "Arapahoe" and "Denver" by a comma and space, we need to add them to the end of the county name in the write() function as follows:

# Write three counties to the file.

txt\_file.write("Arapahoe, ")

txt\_file.write("Denver, ")

txt\_file.write("Jefferson")

The second method is adding all three counties to one line, like this:

# Write three counties to the file.

txt\_file.write("Arapahoe, Denver, Jefferson")

After editing the code and executing one of these two options, open election\_analysis.txt. You'll see that the counties are separated by a comma and a space.

If we want to write each county on a separate line, we need to add the newline escape sequence to the end of each county name. The **newline escape sequence** is the letter "n" preceded by the backward slash like this: \n.

**NOTE**

The newline escape sequence will create a newline, like pressing "return" when it is read. Everything after the \n will be on the next line.

Add the newline escape sequence to the end of the first two county names so that the code looks like this:

# Write three counties to the file.

txt\_file.write("Arapahoe\nDenver\nJefferson")

Edit the code and execute the PyPoll.py file. Then open election\_analysis.txt to see that the counties on separate lines.

**SKILL DRILL**

Modify your code so that the output file looks like this:

End of text box.

Congratulations—you now know how to read a CSV file and write to a text file using Python! This is a huge step because accessing data on a file and writing data to a file are common tasks that programmers and analysts perform.

[Previous](https://courses.bootcampspot.com/courses/1225/modules/items/497189)[Next](https://courses.bootcampspot.com/courses/1225/modules/items/497198)

© 2020 - 2022 Trilogy Education Services, a 2U, Inc. brand. All Rights Reserved.

[Bootcamp: UCD-VIRT-DATA-PT-03-2022-U-B-MW](https://courses.bootcampspot.com/courses/1225)

[3.4.4: Read the Election Results](https://courses.bootcampspot.com/courses/1225)

Immersive Reader

**3.4.4**

**Read the Election Results**

**Tom** tells you that you have been given the green light to read the election results. You know how to open the file, so you can use that code to start. Then, Tom will guide you in reading the data in the election\_results.csv file.

Now it's time to read the election\_results.csv file. As a reminder, the code in our PyPoll.py file should look similar to this code:

# Add our dependencies.

import csv

import os

# Assign a variable to load a file from a path.

file\_to\_load = os.path.join("Resources", "election\_results.csv")

# Assign a variable to save the file to a path.

file\_to\_save = os.path.join("analysis", "election\_analysis.txt")

# Open the election results and read the file.

with open(file\_to\_load) as election\_data:

# To do: read and analyze the data here.

Next, we'll use the reader function to read election\_data.csv.

**IMPORTANT**

Don't forget to add a colon after the with statement and indent on the next line. Forgetting to add a colon will result in a SyntaxError, and not indenting on the next line will result in an IndentationError. In both cases, your code will not run.

**REWIND**

Remember that we found the reader function within the csv module that will read the CSV file.

To PyPoll.py, add the following code where it says # To do: read and analyze the data here. This will allow us to read the CSV file using the csv module with the reader function.

# Read the file object with the reader function.

file\_reader = csv.reader(election\_data)

The variable, file\_reader, is referencing the file object, which is stored in memory. To "pull" the data out of the file object, we can iterate through the file\_reader variable and print each row, including the headers, or column names.

Add the following code to PyPoll.py and run the file in the VS Code terminal.

# Print each row in the CSV file.

for row in file\_reader:

print(row)

Wow—the output was printed quickly! If you blinked, then you might have missed it. So, run it again. If your output for the last 10 lines looks like this, then your code is correct.

While this output was being generated, two things happened:

1. We did not see the headers or columns printed because the output was generated very quickly.
2. Each row in the CSV file was printed out as a list.

In Python, we can retrieve just the headers from the CSV file using a specific method, which we will use a bit later. We can use list indexing to get each element, or ballot ID, county, and candidate, in each row or list.

For our analysis of the election data, we do not need the column headers. Therefore, when we retrieve the data from the CSV file, we will have to skip the first row—the header row.

To skip the header row of the CSV file, use the next() method. This method will skip the first row and return the next item in the list.

Inside the parentheses of the next() method, add the variable file\_reader that is referencing the file object assigned to the CSV file: next(file\_reader).

Running this code will skip the first instance of what is being read. In our case, it is the first row, or header row, of the CSV file. Then we can start iterating through the data starting with the second row.

Just to make sure we are skipping the header row, let's print out the headers for the CSV file. Modify the code as follows and run the file.

# Read the file object with the reader function.

file\_reader = csv.reader(election\_data)

# Print the header row.

headers = next(file\_reader)

print(headers)

At this point, PyPoll.py should look like this:

# Add our dependencies.

import csv

import os

# Assign a variable to load a file from a path.

file\_to\_load = os.path.join("Resources", "election\_results.csv")

# Assign a variable to save the file to a path.

file\_to\_save = os.path.join("analysis", "election\_analysis.txt")

# Open the election results and read the file.

with open(file\_to\_load) as election\_data:

file\_reader = csv.reader(election\_data)

# Read and print the header row.

headers = next(file\_reader)

print(headers)

When you run this code in your VS Code terminal, the output should only show the headers from the CSV file.

['Ballot ID', 'County', 'Candidate']

Now that we have confirmed that we skipped the header row, we can iterate through each row and gather data for our analysis.

**NOTE**

For more information, see the [documentation on using the next() method with the csv module (Links to an external site.)](https://docs.python.org/3.7/library/csv.html#reader-objects).

[Previous](https://courses.bootcampspot.com/courses/1225/modules/items/497195)[Next](https://courses.bootcampspot.com/courses/1225/modules/items/497202)

© 2020 - 2022 Trilogy Education Services, a 2U, Inc. brand. All Rights Reserved.

[Bootcamp: UCD-VIRT-DATA-PT-03-2022-U-B-MW](https://courses.bootcampspot.com/courses/1225)

[3.4.5: Commit Your Code](https://courses.bootcampspot.com/courses/1225)

Immersive Reader

**3.4.5**

# Commit Your Code

**Congratulations!** You got a lot done today. Before moving on, take a moment to commit your code to GitHub. Committing early and often is the best practice to save your work on projects. Plus, Seth and Tom said they might look over your work before tomorrow, so you'll want to update your repository and email them the link, if you have not done so yet.

Save PyPoll.py to the Election\_Analysis folder. To commit your code to your GitHub repository, follow the steps associated with your operating system.

Check out the macOS instructions below, or jump to the [Windows instructions](https://courses.bootcampspot.com/courses/1225/pages/3-dot-4-5-commit-your-code?module_item_id=497202#windows).

## macOS

1. Launch the terminal.
2. Navigate to the Election\_Analysis folder. Here's what the command line might look like:

Toms-MBP:Election\_Analysis tom$

1. Type git status and press Enter. Your output should look like the following:
2. On branch main
3. Your branch is up to date with 'origin/main'.
4. Untracked files:
5. (use "git add <file>..." to include in what will be committed)
6. PyPoll.py

nothing added to commit but untracked files present (use "git add" to track)

**NOTE**

git status tells us the "status" of the GitHub repository. Right now, there is one untracked file, PyPoll.py. The file that needs to be added is in red because it is untracked, meaning it hasn't been added to GitHub to be tracked. When we make changes to the file, these changes are tracked and given a reference number. This is how GitHub keeps track of your files.

1. Add the PyPoll.py file by using git add . and press Enter.
2. Check the status again with git status and press Enter. This ensures that no untracked files have been added.
3. On branch main
4. Your branch is up to date with 'origin/main'.
5. Your branch is up to date with 'origin/main'.
6. Changes to be committed:
7. (use "git reset HEAD <file>..." to unstage)

new file: PyPoll.py

The PyPoll.py file is now green, which means it's being tracked, but it has not been added to your GitHub repository.

1. Commit the files to be added to the repository with git commit -m "Adding PyPoll.py file." and press Enter. The output should look like this:
2. [main aaad131] Adding PyPoll.py file.
3. 1 file changed, 35 insertions(+)

create mode 100644 PyPoll.py

git commit commits the file to be added. -m means "message". After the -m,  add a message in single or double quotes that describes the change to the file, such as "updating PyPoll.py file". After the file is committed, we have to tell the computer to add it to GitHub.

1. Type git push and press Enter. This adds PyPoll.py to the repository. Your output should look something like this:
2. Enumerating objects: 4, done.
3. Counting objects: 100% (4/4), done.
4. Delta compression using up to 8 threads
5. Compressing objects: 100% (3/3), done.
6. Writing objects: 100% (3/3), 777 bytes | 777.00 KiB/s, done.
7. Total 3 (delta 0), reused 0 (delta 0)
8. To https://github.com/<your\_GitHub\_account>/Election\_Analysis.git

15501c0..aaad131 main -> main

1. Refresh your GitHub page to see the changes to your repository.

## Windows

1. Launch Git Bash.
2. Navigate to your Election\_Analysis folder. Here's what the command line should look like:
3. tom@TOM MINGW32 ~/Class/Election\_Analysis (main)

$

In Git Bash, you will know that you are in a tracked GitHub repository when the folder has (main) after the folder name.

1. Type git status and press Enter. Your output should look like this:
2. On branch main
3. Your branch is up to date with 'origin/main'.
4. Untracked files:
5. (use "git add <file>..." to include in what will be committed)
6. PyPoll.py

nothing added to commit but untracked files present (use "git add" to track)

**NOTE**

git status tells us the "status" of the GitHub repository. Right now, there is one untracked file, PyPoll.py. The file that needs to be added is in red because it is untracked, meaning it hasn't been added to GitHub to be tracked. When we make changes to the file, these changes are tracked and given a reference number. This is how GitHub keeps track of your files.

1. Add the PyPoll.py file by using git add . and press Enter.
2. Check the status again with git status and press Enter. This ensures that no untracked files have been added.
3. On branch main
4. Your branch is up to date with 'origin/main'.
5. Your branch is up to date with 'origin/main'.
6. Changes to be committed:
7. (use "git reset HEAD <file>..." to unstage)

new file: PyPoll.py

The PyPoll.py file is now green, which means it's being tracked, but it has not been added to your GitHub repository.

1. Commit the files to be added to the repository with git commit -m "Adding PyPoll.py file." and press Enter. The output should look like this:
2. [main aaad131] Adding PyPoll.py file.
3. 1 file changed, 35 insertions(+)

create mode 100644 PyPoll.py

git commit commits the file to be added. -m means "message". After the -m,  add a message in single or double quotes that describes the change to the file, such as "updating PyPoll.py file". After the file is committed, we have to tell the computer to add it to GitHub.

1. Type git push and press Enter. This adds PyPoll.py to the repository. Your output should look something like this:
2. Enumerating objects: 4, done.
3. Counting objects: 100% (4/4), done.
4. Delta compression using up to 8 threads
5. Compressing objects: 100% (3/3), done.
6. Writing objects: 100% (3/3), 777 bytes | 777.00 KiB/s, done.
7. Total 3 (delta 0), reused 0 (delta 0)
8. To https://github.com/<your\_GitHub\_account>/Election\_Analysis.git

15501c0..aaad131 main -> main

1. Refresh your GitHub page to see the changes to your repository.

Congratulations on adding PyPoll.py to the repository! If you did this without any problems, then give yourself a big pat on the back. But don't worry if you experienced some obstacles; GitHub can be challenging to learn and use at first. With more practice you'll get the hang of it.

[Previous](https://courses.bootcampspot.com/courses/1225/modules/items/497198)[Next](https://courses.bootcampspot.com/courses/1225/modules/items/497210)

© 2020 - 2022 Trilogy Education Services, a 2U, Inc. brand. All Rights Reserved.

[Bootcamp: UCD-VIRT-DATA-PT-03-2022-U-B-MW](https://courses.bootcampspot.com/courses/1225)

[3.5.1: Get the Total Votes](https://courses.bootcampspot.com/courses/1225)

Immersive Reader

**3.5.1**

**Get the Total Votes**

**It's** go time! Using the election data, you will need to determine the total number of votes cast in the election. Using the code from the previous day, Tom will show you how to programmatically count up all the votes cast in the election by amending the code below the for loop.

Open PyPoll.py if the file isn't open already. With the election\_results.csv file open to read the data in our script, we're going to write some code to add up all the votes cast in the election.

Your PyPoll.py file should look similar to the following:

# Add our dependencies.

import csv

import os

# Assign a variable to load a file from a path.

file\_to\_load = os.path.join("Resources", "election\_results.csv")

# Assign a variable to save the file to a path.

file\_to\_save = os.path.join("analysis", "election\_analysis.txt")

# Open the election results and read the file.

with open(file\_to\_load) as election\_data:

file\_reader = csv.reader(election\_data)

# Read the header row.

headers = next(file\_reader)

# Print each row in the CSV file.

for row in file\_reader:

print(row)

To count up all the votes, we need to initialize a variable, which is called an **accumulator**, that will increment by 1 as we read each row in the for loop. For convenience, we will initialize a variable called total\_votes to zero.

The total\_votes variable needs to be placed above the code where we open the file, using the with open() statement. We do this because every time we run the file, the total\_votes variable must be set equal to zero.

After we read the headers, we can iterate through each row and increment the total\_votes variable by 1. The standard Python format to increment a variable is number = number + 1, which can be augmented to number += 1.

In the PyPoll.py file, do the following:

1. Add the total vote counter before the with open() statement.
2. Increment the total\_votes by 1 after the for loop.
3. Print out the total votes.

# Add our dependencies.

import csv

import os

# Assign a variable to load a file from a path.

file\_to\_load = os.path.join("Resources", "election\_results.csv")

# Assign a variable to save the file to a path.

file\_to\_save = os.path.join("analysis", "election\_analysis.txt")

# 1. Initialize a total vote counter.

total\_votes = 0

# Open the election results and read the file

with open(file\_to\_load) as election\_data:

file\_reader = csv.reader(election\_data)

# Read the header row.

headers = next(file\_reader)

# Print each row in the CSV file.

for row in file\_reader:

# 2. Add to the total vote count.

total\_votes += 1

# 3. Print the total votes.

print(total\_votes)

Save the file and run it in the VS Code terminal.

**FINDING**

The total votes should be equal to the total number of rows in election\_results.csv without the header: 369,711.

**REWIND**

The last row number in the CSV file is 369,712, which includes the header.

[Previous](https://courses.bootcampspot.com/courses/1225/modules/items/497202)[Next](https://courses.bootcampspot.com/courses/1225/modules/items/497215)

© 2020 - 2022 Trilogy Education Services, a 2U, Inc. brand. All Rights Reserved.

[Bootcamp: UCD-VIRT-DATA-PT-03-2022-U-B-MW](https://courses.bootcampspot.com/courses/1225)

[3.5.2: Get the Candidates in the Election](https://courses.bootcampspot.com/courses/1225)

Immersive Reader

**3.5.2**

**Get the Candidates in the Election**

**Great** job getting the total votes in the election! Next, you will retrieve the names of the individual candidates in the election. This is important, because later you will need to determine how many votes each candidate received and their percentage of the total vote. So go grab some coffee (or your beverage of choice) and get ready to write some code with Tom.

Remember that when we inspected the data in election\_results.csv, using the first 10 rows and the last 10 rows, there were at two candidate names: Charles Casper Stockham and Raymon Anthony Doane.

**REWIND**

Here's what we found when we looked at the first 10 and last 10 rows of election\_results.csv:

**First 10 RowsLast 10 Rows**

You may have scrolled through the CSV file using Excel or VS Code and found three candidate's names. However, this method isn't very efficient, as it takes a long time to scroll through 369,712 rows. With Python, we can iterate through the rows in the CSV file and get the candidates from the "Candidate" column, and then add their names to a list.

**REWIND**

Remember, when we iterated through the rows of election\_results.csv, the last 10 rows that were printed to the terminal had the structure of Python list.

To get the candidate from each list when we iterate through the row, we can use indexing on the for loop variable, row. The Candidate column is the third column that has the second index, so we would use, row[2] to reference the Candidate column.

Let's test this to make sure. Follow these steps:

1. Declare a new list, candidate\_options = [] by adding it before the with open() statement in our script.
2. Add the following code to get the candidate's name from the row within the for loop.
3. # Print the candidate name from each row

candidate\_name = row[2]

1. Add the candidate\_name to the candidate\_options list using the append() method.

**REWIND**

To add an item to a list, use the append() method.

1. Add a print statement that is flush with the left margin to print out the candidate\_options list.

Your file should look like this:

# Add our dependencies.

import csv

import os

# Assign a variable to load a file from a path.

file\_to\_load = os.path.join("Resources", "election\_results.csv")

# Assign a variable to save the file to a path.

file\_to\_save = os.path.join("analysis", "election\_analysis.txt")

# Initialize a total vote counter.

total\_votes = 0

# Candidate Options

candidate\_options = []

# Open the election results and read the file.

with open(file\_to\_load) as election\_data:

file\_reader = csv.reader(election\_data)

# Read the header row.

headers = next(file\_reader)

# Print each row in the CSV file.

for row in file\_reader:

# Add to the total vote count.

total\_votes += 1

# Print the candidate name from each row.

candidate\_name = row[2]

# Add the candidate name to the candidate list.

candidate\_options.append(candidate\_name)

# Print the candidate list.

print(candidate\_options)

When we run this file, we will see all the elements, or candidates' names, from each row in the candidate\_options list:

However, we do not want every candidate from each row. Instead, we need to get only the unique candidate names.

To get the unique names in the candidate\_options list, we can use an if statement with the not in membership operator to check if the candidate has been added to the list. If the candidate's name has been added to the list, then the next time the candidate's name is found in a row using the for loop, it will not be added to the list.

**HINT**

Inside the for loop, we will need to check if the candidate has been added to the candidate\_options list. Therefore, add the following code to PyPoll.py inside the for loop.

# If the candidate does not match any existing candidate...

if candidate\_name not in candidate\_options:

# Add it to the list of candidates.

candidate\_options.append(candidate\_name)

The code in the for loop should look like this:

# Print each row in the CSV file.

for row in file\_reader:

# Add to the total vote count.

total\_votes += 1

# Print the candidate name from each row.

candidate\_name = row[2]

# If the candidate does not match any existing candidate...

if candidate\_name not in candidate\_options:

# Add it to the list of candidates.

candidate\_options.append(candidate\_name)

# Print the candidate list.

print(candidate\_options)

Run the file in the VS Code terminal.

**FINDING**

The output will be a list of the candidates in the election.

['Charles Casper Stockham', 'Diana DeGette', 'Raymon Anthony Doane']

[Previous](https://courses.bootcampspot.com/courses/1225/modules/items/497210)[Next](https://courses.bootcampspot.com/courses/1225/modules/items/497221)

© 2020 - 2022 Trilogy Education Services, a 2U, Inc. brand. All Rights Reserved.

[Bootcamp: UCD-VIRT-DATA-PT-03-2022-U-B-MW](https://courses.bootcampspot.com/courses/1225)

[3.5.3: Get the Candidates' Votes](https://courses.bootcampspot.com/courses/1225)

Immersive Reader

**3.5.3**

# Get the Candidates' Votes

**Seth** is really happy with the work you're doing. Even though you have written a lot of code with Tom's guidance, you're becoming a programmer and analyst in your own right. But there's still work to be done! Now that you have created the if statement to get the unique names of the candidates list, the next task is to count the votes for each candidate in the if statement as you iterate through the rows of the CSV file.

In the last section, we focused on finding the total number of votes cast in the election as well as the names of the candidates. Now the goal is to find the number of votes cast for each candidate.

We can count the votes for each candidate while we are in the if statement. As we iterate through each row of the CSV file, we can increment the votes for each candidate by 1. However, we need to link those votes with a candidate. A convenient way to do this is to create a dictionary where the key is each candidate's name and the vote count for that candidate is the value for the key.

The structure of the dictionary may look similar to this:

candidate\_votes = {"candidate\_name1": votes, "candidate\_name2": votes, "candidate\_name3": votes}

To create this dictionary, we first need to declare an empty dictionary, candidate\_votes = {}, as we did with the empty candidate\_options list. The dictionary will need to be placed before the with open() statement as we did with the candidate\_options list. Follow these steps.

First, declare an empty dictionary, candidate\_votes = {} in the PyPoll.py file.

# Add our dependencies.

import csv

import os

# Assign a variable to load a file from a path.

file\_to\_load = os.path.join("Resources", "election\_results.csv")

# Assign a variable to save the file to a path.

file\_to\_save = os.path.join("analysis", "election\_analysis.txt")

# Initialize a total vote counter.

total\_votes = 0

# Candidate options and candidate votes

candidate\_options = []

# 1. Declare the empty dictionary.

candidate\_votes = {}

# Open the election results and read the file.

with open(file\_to\_load) as election\_data:

Inside the if statement, we need to instantiate a candidate as a key for the dictionary. In other words, we need to create a key from the unique candidates.

**REWIND**

We used the format dictionary\_name[key] to get the value for the key. We can use this same format to create a key.

To create each candidate as a key, use candidate\_votes[candidate\_name].

Add candidate\_votes[candidate\_name] = 0 inside the if statement.

# Open the election results and read the file.

with open(file\_to\_load) as election\_data:

file\_reader = csv.reader(election\_data)

# Read the header row.

headers = next(file\_reader)

# Print each row in the CSV file.

for row in file\_reader:

# Add to the total vote count.

total\_votes += 1

# Print the candidate name from each row.

candidate\_name = row[2]

if candidate\_name not in candidate\_options:

# Add the candidate name to the candidate list.

candidate\_options.append(candidate\_name)

# 2. Begin tracking that candidate's vote count.

candidate\_votes[candidate\_name] = 0

# Print the candidate vote dictionary.

print(candidate\_votes)

When we add candidate\_votes[candidate\_name] = 0, we're setting each candidate's vote count to zero. Once we set it to zero, then we can start tallying the votes for each candidate.

The PyPoll.py file should look like the following:

# Add our dependencies.

import csv

import os

# Assign a variable to load a file from a path.

file\_to\_load = os.path.join("Resources", "election\_results.csv")

# Assign a variable to save the file to a path.

file\_to\_save = os.path.join("analysis", "election\_analysis.txt")

# Initialize a total vote counter.

total\_votes = 0

# Candidate options and candidate votes

candidate\_options = []

# 1. Declare the empty dictionary.

candidate\_votes = {}

# Open the election results and read the file.

with open(file\_to\_load) as election\_data:

file\_reader = csv.reader(election\_data)

# Read the header row.

headers = next(file\_reader)

# Print each row in the CSV file.

for row in file\_reader:

# Add to the total vote count.

total\_votes += 1

# Print the candidate name from each row.

candidate\_name = row[2]

if candidate\_name not in candidate\_options:

# Add the candidate name to the candidate list.

candidate\_options.append(candidate\_name)

# 2. Begin tracking that candidate's vote count.

candidate\_votes[candidate\_name] = 0

# Print the candidate vote dictionary.

print(candidate\_votes)

When we run the file, the output is each candidate's name as the key and the candidate's vote count as the value. At the moment, each candidate has zero votes!

{'Raymon Anthony Doane': 0, 'Diana DeGette': 0, 'Charles Casper Stockham': 0}

So, let's add votes to each candidate. Using the Python format for incrementing variables, we'll increment each candidate\_votes[candidate\_name] every time that name appears while we are iterating through each row.

**REWIND**

The standard Python format to increment a variable is number = number + 1, which can be augmented to number += 1.

To begin tracking the candidate's vote count, we initialize each candidate's vote equal to zero. Next, we need to increment the votes by 1 every time a candidate name appears in a row. Incrementing the votes for each candidate inside the if statement will increment the candidate's vote by only 1 every time we run the file.

Let's test this concept. Add the following code in PyPoll.py, inside the if statement.

candidate\_votes[candidate\_name] += 1

Your code should look like this inside the if statement:

if candidate\_name not in candidate\_options:

# Add the candidate name to the candidate list.

candidate\_options.append(candidate\_name)

# Begin tracking that candidate's vote count.

candidate\_votes[candidate\_name] = 0

# Add a vote to that candidate's count.

candidate\_votes[candidate\_name] += 1

# Print the candidate vote dictionary.

print(candidate\_votes)

When we run the file, each candidate has only one vote.

{'Raymon Anthony Doane': 1, 'Diana DeGette': 1, 'Charles Casper Stockham': 1}

This will be the output every time we run this file. We need to fix this so that each candidate's vote count is incremented as we iterate through each row.

To increment each candidate's vote count every time their name appears in a row, we need to move the vote counter inside the for loop and have it align with the if statement, like this:

if candidate\_name not in candidate\_options:

# Add the candidate name to the candidate list.

candidate\_options.append(candidate\_name)

# Begin tracking that candidate's vote count.

candidate\_votes[candidate\_name] = 0

# Add a vote to that candidate's count

candidate\_votes[candidate\_name] += 1

# Print the candidate vote dictionary.

print(candidate\_votes)

After moving the vote counter, run the file to get each candidate's vote count.

**FINDING**

The output shows each candidate and their vote count:

{'Charles Casper Stockham': 85213, 'Diana DeGette': 272892, 'Raymon Anthony Doane': 11606}

[Previous](https://courses.bootcampspot.com/courses/1225/modules/items/497215)[Next](https://courses.bootcampspot.com/courses/1225/modules/items/497225)

© 2020 - 2022 Trilogy Education Services, a 2U, Inc. brand. All Rights Reserved.

[Bootcamp: UCD-VIRT-DATA-PT-03-2022-U-B-MW](https://courses.bootcampspot.com/courses/1225)

[3.5.4: Determine Candidates' Percentage of Votes](https://courses.bootcampspot.com/courses/1225)

Immersive Reader

**3.5.4**

**Determine Candidates' Percentage of Votes**

**With** the total votes and each candidate's votes in hand, you can now calculate the percentage of votes each candidate received. You feel confident doing this because you've been doing so performing mathematical operations.

Now that we have the total number of votes as well as the number of votes each candidate received, our next task is to calculate the percentage of votes for each candidate. This means we need to divide the candidate's vote count by the total vote count, and then multiply by 100. Here's the equation:

vote\_percentage = (votes / total\_votes) \* 100

The votes are the values of each candidate\_name in the candidate\_votes dictionary. To get a percentage, we need to convert votes and total\_votes to floating-point decimal numbers because the votes in the dictionary and the total\_votes are integers.

To retrieve the votes for each candidate and get the percentage of votes, follow these steps:

1. Use a for loop to iterate through the candidate\_options = [] list. We will get the candidate's name.
2. Use the for loop variable to retrieve the votes of the candidate from the candidate\_votes = {} dictionary.
3. Calculate the percentage of the vote count.
4. Print each candidate and the percentage of votes using f-string formatting.

Add the following code below to your script after candidate\_votes[candidate\_name] += 1 and make it flush with the left margin. Here's how this code might look.

# Determine the percentage of votes for each candidate by looping through the counts.

# 1. Iterate through the candidate list.

for candidate\_name in candidate\_votes:

# 2. Retrieve vote count of a candidate.

votes = candidate\_votes[candidate\_name]

# 3. Calculate the percentage of votes.

vote\_percentage = float(votes) / float(total\_votes) \* 100

# 4. Print the candidate name and percentage of votes.

print(f"{candidate\_name}: received {vote\_percentage}% of the vote.")

Save and run the file.

**FINDING**

After executing the code, the PyPoll.py output will be:

Charles Casper Stockham: received 23.04854332167558% of the vote

Diana DeGette: received 73.81224794501652% of the vote.

Raymon Anthony Doane: received 3.1392087333079077% of the vote

**SKILL DRILL**

Modify the f-string print statement in Step 4 (print the candidate name and percentage of votes) so the percentage is formatted to one decimal place.

End of text box.

[Previous](https://courses.bootcampspot.com/courses/1225/modules/items/497221)[Next](https://courses.bootcampspot.com/courses/1225/modules/items/497230)

© 2020 - 2022 Trilogy Education Services, a 2U, Inc. brand. All Rights Reserved.

[Bootcamp: UCD-VIRT-DATA-PT-03-2022-U-B-MW](https://courses.bootcampspot.com/courses/1225)

[3.5.5: Determine the Winning Candidate](https://courses.bootcampspot.com/courses/1225)

Immersive Reader

**3.5.5**

**Determine the Winning Candidate**

**You** have now helped Tom find all the candidates' names, the number of votes that each candidate received, and the percentage of votes for each candidate. Now Seth would like Tom to walk you through how to determine the winning candidate, vote count, and percentage. For this you will need to use a decision statement to compare the number of votes each candidate received.

We have tabulated all the votes and calculated the vote percentages. Your final task is to determine the winning candidate by the number and percentage of votes.

When we loop through the vote counts, we can:

* Use an if statement to check if the first vote count for a candidate is greater than zero.
* If the statement is true, then that vote count will be equal to the "winning count."
* At the same time, we can set that candidate's percentage of the vote equal to the "winning percentage."
* Then we can select the candidate as the "winning candidate" from the candidate\_options list.

To do all of this, we will first need to:

* Declare a variable that holds an empty string value for the winning candidate.
* Declare a variable for the "winning count" equal to zero.
* Declare a variable for the "winning\_percentage" equal to zero.

Add the following code before the with open() statement:

# Winning Candidate and Winning Count Tracker

winning\_candidate = ""

winning\_count = 0

winning\_percentage = 0

Next, we will create an if statement inside the for loop and do the following:

* Determine if the vote count that was calculated is greater than the winning\_count.
* If the vote count is greater than the winning\_count and the percentage is greater than the winning\_percentage, set the winning\_count equal to the votes and set the winning\_percentage equal to the vote\_percentage.
* Set the winning\_count equal to the variable, candidate\_name, in the for loop.

Add the following code inside the for loop where we iterated through the candidate\_options list.

# Determine winning vote count and candidate

# 1. Determine if the votes are greater than the winning count.

if (votes > winning\_count) and (vote\_percentage > winning\_percentage):

# 2. If true then set winning\_count = votes and winning\_percent =

# vote\_percentage.

winning\_count = votes

winning\_percentage = vote\_percentage

# 3. Set the winning\_candidate equal to the candidate's name.

winning\_candidate = candidate\_name

Your code should look like this:

# Determine the percentage of votes for each candidate by looping through the counts.

# Iterate through the candidate list.

for candidate\_name in candidate\_votes:

# Retrieve vote count of a candidate.

votes = candidate\_votes[candidate\_name]

# Calculate the percentage of votes.

vote\_percentage = float(votes) / float(total\_votes) \* 100

# To do: print out each candidate's name, vote count, and percentage of

# votes to the terminal.

# Determine winning vote count and candidate

# Determine if the votes is greater than the winning count.

if (votes > winning\_count) and (vote\_percentage > winning\_percentage):

# If true then set winning\_count = votes and winning\_percent =

# vote\_percentage.

winning\_count = votes

winning\_percentage = vote\_percentage

# And, set the winning\_candidate equal to the candidate's name.

winning\_candidate = candidate\_name

# To do: print out the winning candidate, vote count and percentage to

# terminal.

Finally, we will add a print statement that prints each candidate's name, their vote count, and their percentage of votes. We'll also add a print statement that prints the winning candidate, winning vote count, and winning percentage.

Add the following code **before** the if statement, where it says # To do: print out each candidate's name, vote count, and percentage of votes to the terminal. To print out each candidate's name, vote count, and percentage of votes while we check the vote on a newline.

# To do: print out each candidate's name, vote count, and percentage of

# votes to the terminal.

print(f"{candidate\_name}: {vote\_percentage:.1f}% ({votes:,})\n")

If we run the file, the output will be:

Charles Casper Stockham: 23.0% (85,213)

Diana DeGette: 73.8% (272,892)

Raymon Anthony Doane: 3.1% (11,606)

Next, add the following code after the if statement and aligned with the for loop, for candidate in candidate\_votes, to print out the winning candidate summary.

winning\_candidate\_summary = (

f"-------------------------\n"

f"Winner: {winning\_candidate}\n"

f"Winning Vote Count: {winning\_count:,}\n"

f"Winning Percentage: {winning\_percentage:.1f}%\n"

f"-------------------------\n")

print(winning\_candidate\_summary)

Run the file. Your output should have printed out the following results:

Charles Casper Stockham: 23.0% (85,213)

Diana DeGette: 73.8% (272,892)

Raymon Anthony Doane: 3.1% (11,606)

-------------------------

Winner: Diana DeGette

Winning Vote Count: 272,892

Winning Percentage: 73.8%

-------------------------

Congratulations on completing the election audit!

**FINDING**

Your results should tell you that Diane DeGette was the winner of the election with 73.8% of the vote and 272,892 votes.

[Previous](https://courses.bootcampspot.com/courses/1225/modules/items/497225)[Next](https://courses.bootcampspot.com/courses/1225/modules/items/497234)

© 2020 - 2022 Trilogy Education Services, a 2U, Inc. brand. All Rights Reserved.

[Bootcamp: UCD-VIRT-DATA-PT-03-2022-U-B-MW](https://courses.bootcampspot.com/courses/1225)

[3.5.6: Commit Your Code](https://courses.bootcampspot.com/courses/1225)

Immersive Reader

**3.5.6**

**Commit Your Code**

**Congratulations** on completing the election audit! You got a lot done today. Before moving on, take a moment to commit your code to GitHub. Seth and Tom are going to look over your work before finalizing the election audit, so you will need to update your repository.

Save the PyPoll.py file to your Election\_Analysis folder. To commit to GitHub, follow the steps for your operating system.

1. Launch the terminal for macOS or GitBash for Windows.
2. Navigate to the Election\_Analysis folder.
3. Type git status and press Enter.
4. Type git add . and press Enter.
5. Check the status again with git status and then press Enter.
6. Commit the files to be added to the repository by typing git commit -m "Adding the election audit code." and then press Enter.
7. Type git push and press Enter to add the file to your repository.
8. Refresh your GitHub page to see the repository changes.

[Previous](https://courses.bootcampspot.com/courses/1225/modules/items/497230)[Next](https://courses.bootcampspot.com/courses/1225/modules/items/497243)

© 2020 - 2022 Trilogy Education Services, a 2U, Inc. brand. All Rights Reserved.

[Bootcamp: UCD-VIRT-DATA-PT-03-2022-U-B-MW](https://courses.bootcampspot.com/courses/1225)

[3.6.1: Write the Election Results to a Text File](https://courses.bootcampspot.com/courses/1225)

Immersive Reader

**3.6.1**

**Write the Election Results to a Text File**

**Seth** and Tom have looked over your election audit and are pleased with what you have accomplished. Now, Seth needs you to save your election audit results to a text file so he can send it to the election commission. Don't worry if you don't remember how to write data to file—Tom is going to walk you through the process to refresh your memory.

We need to save the election results to a text file and then print the file to the screen to make sure that the results are in the correct format.

After the election analysis is written to the text file, the file should look like this:

Now code should look like this:

# Add our dependencies.

import csv

import os

# Assign a variable to load a file from a path.

file\_to\_load = os.path.join("Resources", "election\_results.csv")

# Assign a variable to save the file to a path.

file\_to\_save = os.path.join("analysis", "election\_analysis.txt")

# Initialize a total vote counter.

total\_votes = 0

# Candidate options and candidate votes

candidate\_options = []

candidate\_votes = {}

# Track the winning candidate, vote count, and percentage.

winning\_candidate = ""

winning\_count = 0

winning\_percentage = 0

# Open the election results and read the file.

with open(file\_to\_load) as election\_data:

file\_reader = csv.reader(election\_data)

# Read the header row.

headers = next(file\_reader)

# Print each row in the CSV file.

for row in file\_reader:

# Add to the total vote count.

total\_votes += 1

# Get the candidate name from each row.

candidate\_name = row[2]

# If the candidate does not match any existing candidate add it the

# the candidate list.

if candidate\_name not in candidate\_options:

# Add the candidate name to the candidate list.

candidate\_options.append(candidate\_name)

# And begin tracking that candidate's voter count.

candidate\_votes[candidate\_name] = 0

# Add a vote to that candidate's count

candidate\_votes[candidate\_name] += 1

for candidate\_name in candidate\_votes:

# Retrieve vote count and percentage.

votes = candidate\_votes[candidate\_name]

vote\_percentage = float(votes) / float(total\_votes) \* 100

# Print each candidate, their voter count, and percentage to the

# terminal.

print(f"{candidate\_name}: {vote\_percentage:.1f}% ({votes:,})\n")

# Determine winning vote count, winning percentage, and candidate.

if (votes > winning\_count) and (vote\_percentage > winning\_percentage):

winning\_count = votes

winning\_candidate = candidate\_name

winning\_percentage = vote\_percentage

# Print the winning candidates' results to the terminal.

winning\_candidate\_summary = (

f"-------------------------\n"

f"Winner: {winning\_candidate}\n"

f"Winning Vote Count: {winning\_count:,}\n"

f"Winning Percentage: {winning\_percentage:.1f}%\n"

f"-------------------------\n")

print(winning\_candidate\_summary)

Now we are going to modify our code so we can write the election\_results to a text file.

First, comment out print(f"{candidate\_name}: {vote\_percentage:.1f}% ({votes:,})\n") and print(winning\_candidate\_summary) by adding a # in front of both lines.

Next, insert with open(file\_to\_save, "w") as txt\_file: after candidate\_votes[candidate\_name] += 1. Make sure that the filename file\_to\_save is in the "w" mode to write data to the file.

This section of the code should look like this:

# Add a vote to that candidate's count

candidate\_votes[candidate\_name] += 1

# Save the results to our text file.

with open(file\_to\_save, "w") as txt\_file:

Next, indent all the code below with open(file\_to\_save, "w") as txt\_file: by four spaces. To do this, select all the code and comments below the with open section and press the Tab key once.

Next, below the with open statement, write the text shown on lines 2–5 in the following image as an f-string literal message, "election\_results".

Add this code below the with open statement:

# Print the final vote count to the terminal.

election\_results = (

f"\nElection Results\n"

f"-------------------------\n"

f"Total Votes: {total\_votes:,}\n"

f"-------------------------\n")

print(election\_results, end="")

# Save the final vote count to the text file.

txt\_file.write(election\_results)

Let's examine the code we are writing to the text file file\_to\_save and printing to the terminal.

* The variable, election\_results, has four strings written to it.
* The first string, Election Results, has the newline character, \n, before and after it. When this line is saved to the text file or printed, it will be on the second line; then a newline is created.
* On the third line, 25 dashes will be printed, and then a newline line is created.
* On the fourth line, the string Total Votes: {total\_votes:,} will be printed with the votes formatted with a thousands separator, and then a newline is created.
* On the fifth line, 25 dashes will be printed, and then a newline is created.
* Then, we print the election\_results, with the parameter end="" equal to an empty string.
* Finally, we write election\_results to the text file.

**NOTE**

By default, the "end" parameter will print a newline, \n. You can add something between the double quotes, which will be printed after the 25 dashes are printed to the terminal. The "end" parameter is added to ensure that nothing will be printed on the last line when the election\_results. Anything code that is printed after print(election\_results, end="") will be printed on a newline.

When we execute PyPoll.py, the output to the terminal will look like this:

Election Results

-------------------------

Total Votes: 369,711

-------------------------

The election\_results.txt file will look like this:

[Previous](https://courses.bootcampspot.com/courses/1225/modules/items/497234)[Next](https://courses.bootcampspot.com/courses/1225/modules/items/497249)

© 2020 - 2022 Trilogy Education Services, a 2U, Inc. brand. All Rights Reserved.

[Bootcamp: UCD-VIRT-DATA-PT-03-2022-U-B-MW](https://courses.bootcampspot.com/courses/1225)

[3.6.2: Write the Candidates' Results to a Text File](https://courses.bootcampspot.com/courses/1225)

Immersive Reader

**3.6.2**

# Write the Candidates' Results to a Text File

**Tom** has asked you to add the candidates' results to a text file just below the election results. This will involve some modification to your original script, but don't worry. If you break the code from the original script, you can retrieve the old version from GitHub. But to make sure you don't make any mistakes that break your code, Tom is going to guide your through the editing process.

Now that we have modified the script to print the election results to a text file, we're going to save each candidate's election results to the election\_analysis.txt file.

**REWIND**

The code for printing each candidate's election results is as follows:

print(f"{candidate\_name}: {vote\_percentage:.1f}% ({votes:,})\n")

To add each candidate's election results to the election\_analysis.txt file, we are going to take out the code inside the print() function and place it in a variable, candidate\_results, like this:

candidate\_results = (f"{candidate\_name}: {vote\_percentage:.1f}% ({votes:,})\n")

We can print the candidate\_results to the output terminal.

Below the candidate\_results variable, add the following code to save the candidate\_results to the election\_analysis.txt file, and print the candidate's election results to the terminal.

# Print each candidate, their voter count, and percentage to the terminal.

print(candidate\_results)

# Save the candidate results to our text file.

txt\_file.write(candidate\_results)

The output to the terminal after executing this code will look like this:

Our text file, election\_results.txt, will look like this:

The difference between lines 6–8 in the text file and what you see in the terminal output has to do with when we add a newline character, /n, in the print statement for the candidate's election results.

When we print to the terminal, the print() function will create a newline for the next output. When we add a newline character, /n, in the print statement we are telling Python to add another newline. This is why you see a double-space between the candidates' results in the terminal. In the text file, there is no print statement, so we need to add the newline escape sequence, /n, to print each candidate's election results on a newline.

[Previous](https://courses.bootcampspot.com/courses/1225/modules/items/497243)[Next](https://courses.bootcampspot.com/courses/1225/modules/items/497251)

© 2020 - 2022 Trilogy Education Services, a 2U, Inc. brand. All Rights Reserved.

[Bootcamp: UCD-VIRT-DATA-PT-03-2022-U-B-MW](https://courses.bootcampspot.com/courses/1225)

[3.6.3: Write the Winning Candidate's Results to a Text File](https://courses.bootcampspot.com/courses/1225)

Immersive Reader

**3.6.3**

# Write the Winning Candidate's Results to a Text File

**You're** almost finished! There is one last piece of coding that you need to do: save the winning candidate's results to the election\_results.txt file and print these results to the VS Code output terminal. Tom is going to give you some instructions but then let you to work independently to finish this last piece of coding.

We are almost done! There is one more modification to make to the original script—to save the winning candidate's election results to the election\_results.txt file.

We have already printed the winning\_candidate\_summary to the terminal, so all we have to do is write this variable to the text file.

Add the following code at the end of your file, and make sure it is aligned with the for loop, for candidate in candidate\_votes:

# Save the winning candidate's name to the text file.

txt\_file.write(winning\_candidate\_summary)

The final code should look like this:

# Add our dependencies.

import csv

import os

# Assign a variable to load a file from a path.

file\_to\_load = os.path.join("Resources", "election\_results.csv")

# Assign a variable to save the file to a path.

file\_to\_save = os.path.join("analysis", "election\_analysis.txt")

# Initialize a total vote counter.

total\_votes = 0

# Candidate options and candidate votes.

candidate\_options = []

candidate\_votes = {}

# Track the winning candidate, vote count, and percentage.

winning\_candidate = ""

winning\_count = 0

winning\_percentage = 0

# Open the election results and read the file.

with open(file\_to\_load) as election\_data:

file\_reader = csv.reader(election\_data)

# Read the header row.

headers = next(file\_reader)

# Print each row in the CSV file.

for row in file\_reader:

# Add to the total vote count.

total\_votes += 1

# Get the candidate name from each row.

candidate\_name = row[2]

# If the candidate does not match any existing candidate, add the

# the candidate list.

if candidate\_name not in candidate\_options:

# Add the candidate name to the candidate list.

candidate\_options.append(candidate\_name)

# And begin tracking that candidate's voter count.

candidate\_votes[candidate\_name] = 0

# Add a vote to that candidate's count.

candidate\_votes[candidate\_name] += 1

# Save the results to our text file.

with open(file\_to\_save, "w") as txt\_file:

# After opening the file print the final vote count to the terminal.

election\_results = (

f"\nElection Results\n"

f"-------------------------\n"

f"Total Votes: {total\_votes:,}\n"

f"-------------------------\n")

print(election\_results, end="")

# After printing the final vote count to the terminal save the final vote count to the text file.

txt\_file.write(election\_results)

for candidate\_name in candidate\_votes:

# Retrieve vote count and percentage.

votes = candidate\_votes[candidate\_name]

vote\_percentage = float(votes) / float(total\_votes) \* 100

candidate\_results = (

f"{candidate\_name}: {vote\_percentage:.1f}% ({votes:,})\n")

# Print each candidate's voter count and percentage to the terminal.

print(candidate\_results)

# Save the candidate results to our text file.

txt\_file.write(candidate\_results)

# Determine winning vote count, winning percentage, and winning candidate.

if (votes > winning\_count) and (vote\_percentage > winning\_percentage):

winning\_count = votes

winning\_candidate = candidate\_name

winning\_percentage = vote\_percentage

# Print the winning candidate's results to the terminal.

winning\_candidate\_summary = (

f"-------------------------\n"

f"Winner: {winning\_candidate}\n"

f"Winning Vote Count: {winning\_count:,}\n"

f"Winning Percentage: {winning\_percentage:.1f}%\n"

f"-------------------------\n")

print(winning\_candidate\_summary)

# Save the winning candidate's results to the text file.

txt\_file.write(winning\_candidate\_summary)

When we execute this code, we get the overall election results, each candidate's results, and the winning candidate summary printed to the terminal in VS Code, as well as saved to the election\_results.txt file.

Congratulations on a job well done! This was a long script that required a lot of coding and modification, and you put in a lot of hard work getting up-to-speed in Python. You should feel proud of what you have accomplished.

[Previous](https://courses.bootcampspot.com/courses/1225/modules/items/497249)[Next](https://courses.bootcampspot.com/courses/1225/modules/items/497254)

© 2020 - 2022 Trilogy Education Services, a 2U, Inc. brand. All Rights Reserved.

[Bootcamp: UCD-VIRT-DATA-PT-03-2022-U-B-MW](https://courses.bootcampspot.com/courses/1225)

[3.6.4: Commit the Final Code to GitHub](https://courses.bootcampspot.com/courses/1225)

Immersive Reader

**3.6.4**

**Commit the Final Code to GitHub**

**Congratulations** on finalizing the results! You have done a lot of programming and have both learned and achieved a lot over the course of this audit. Before moving on, commit your code to GitHub. Seth and Tom are going to look over your final piece of work before Seth sends your file to the election commission.

Now that we've completed our analysis, we need to save and commit our changes to GitHub. Save PyPoll.py to the Election\_Analysis folder, and then follow these steps:

1. Launch the terminal for macOS or GitBash for Windows.
2. Navigate to the Election\_Analysis folder using the necessary commands.
3. Type git status and press Enter.
4. Type git add . and press Enter.
5. Check the status again with git status and then press Enter.
6. Commit the files to be added to the repository by typing git commit -m "Adding the election analysis." and then press Enter.
7. Type git push and press Enter to add the file to your repository.
8. Refresh your GitHub page to see the repository changes.

[Previous](https://courses.bootcampspot.com/courses/1225/modules/items/497251)[Next](https://courses.bootcampspot.com/courses/1225/modules/items/497259)

© 2020 - 2022 Trilogy Education Services, a 2U, Inc. brand. All Rights Reserved.

[Bootcamp: UCD-VIRT-DATA-PT-03-2022-U-B-MW](https://courses.bootcampspot.com/courses/1225)

[3.6.5: Update Your README File](https://courses.bootcampspot.com/courses/1225)

Immersive Reader

**3.6.5**

# Update Your README File

**Since** you did a great job on the election audit, Tom and Seth are going to share your GitHub repository with others on the team to showcase your code and results. This would be a good time to update your README.md file on GitHub with a description of your work and the results of the analysis.

Updating your GitHub repository README.md file is important because it allows you to explain the purpose of your repository and summarize key findings in your analysis. An .md file is a **markdown file**, the standard text file that GitHub uses.

**NOTE**

We can edit a markdown file in GitHub directly or use VS Code. Here, we'll be using GitHub to better illustrate the changes that occur when editing a markdown file.

After you push up all of your edited files and folders to your GitHub repository, your repository should look something like this:

To update the README.md file, click the icon that looks like a pencil on the right-hand side.

The screen should now look like this:

Now we can edit the README.md file. It's a best practice to provide at least a project overview, a list of resources and software—including the version of the software you used—and a summary of the findings. The following screenshot shows an example of a descriptive README.md file.

Note that you will need to provide the specific details for the election audit summary.

**NOTE**

You will submit your completed Challenge using a GitHub URL. Be sure to provide a description of the Challenge. When you're done, add a description of your key findings.

When you are finished editing the README, scroll down to the bottom of the page and write a commit message below the "Commit changes" header, in the field that says "commit message here." This is the same message you would type when using the git commit -m command. After adding the message, press the green "Commit changes" button to commit the changes to your repository.

When the changes have been committed, refresh the GitHub repository homepage. You should see the changes made to the README file. To see the changes on your README file in your repository folder on your computer, navigate to your folder, type git pull, and then press Enter.

**NOTE**

For more information, see the [GitHub documentation on basic writing and formatting syntax (Links to an external site.)](https://help.github.com/en/articles/basic-writing-and-formatting-syntax).

[Previous](https://courses.bootcampspot.com/courses/1225/modules/items/497254)[Next](https://courses.bootcampspot.com/courses/1225/modules/items/497268)

© 2020 - 2022 Trilogy Education Services, a 2U, Inc. brand. All Rights Reserved.

[Bootcamp: UCD-VIRT-DATA-PT-03-2022-U-B-MW](https://courses.bootcampspot.com/courses/1225)

[Module 3 Challenge](https://courses.bootcampspot.com/courses/1225)

Immersive Reader

# Module 3 Challenge

Start Assignment

* **Due** Monday by 2:59am

* **Points** 100

* **Submitting** a text entry box or a website url

## Background

Congratulations! You’ve helped Seth and Tom submit the election audit results to the election commission. But wait! The election commission has requested some additional data to complete the audit:

* The voter turnout for each county
* The percentage of votes from each county out of the total count
* The county with the highest turnout

Working from this module’s election\_results.csv file, use for loops and conditional statements with membership and logical operators to find the requested results. Then, print the results to the command line and save them to your election\_results.txt file.

Finally, you’ll provide a written analysis of the election audit for the election commission, including the new results and a clearly written overview of your methods. As with all written analyses, this will help your audience understand what you did and what they might be able to do with the data you presented.

## What You're Creating

This new assignment consists of two technical analysis deliverables and a written report to deliver your results. You will submit the following:

* Deliverable 1: The Election Results Printed to the Command Line
* Deliverable 2: The Election Results Saved to a Text File
* Deliverable 3: A written Analysis of the Election Audit (README.md)

## Files

Use the following link to download the challenge starter code, which includes the Module 3 PyPoll solution.

[Download challenge starter code (Links to an external site.)](https://2u-data-curriculum-team.s3.amazonaws.com/dataviz-online/module_3/PyPoll_Challenge_starter_code.py)

## Deliverable 1: Election Results Printed to the Command Line (50 points)

### Deliverable 1 Instructions

Using repetition statements, conditional statements with logical operators, and print statements, print out the candidate and county election results to the command line.

**REWIND**

For this deliverable, you’ve already done the following in this module:

* [**Lesson 3.2.2:**](https://courses.bootcampspot.com/courses/1225/pages/3-dot-2-2-execute-python-files) Run a Python file in the command line or VS Code.
* [**Lesson 3.2.4:**](https://courses.bootcampspot.com/courses/1225/pages/3-dot-2-4-perform-calculations-using-python) Perform Calculations.
* [**Lesson 3.2.5:**](https://courses.bootcampspot.com/courses/1225/pages/3-dot-2-5-data-structures-lists) Create and add to a list.
* [**Lesson 3.2.7:**](https://courses.bootcampspot.com/courses/1225/pages/3-dot-2-7-data-structures-dictionaries) Create and add keys and values to a dictionary.
* [**Lesson 3.2.8:**](https://courses.bootcampspot.com/courses/1225/pages/3-dot-2-8-decision-statements) Use decision statements to check a condition.
* [**Lesson 3.2.9:**](https://courses.bootcampspot.com/courses/1225/pages/3-dot-2-9-membership-and-logical-operators) Apply membership and logical operators to decision statements.
* [**Lesson 3.2.10:**](https://courses.bootcampspot.com/courses/1225/pages/3-dot-2-10-repetition-statements) Use repetition statements to iterate through a list or dictionary.
* [**Lesson 3.2.11:**](https://courses.bootcampspot.com/courses/1225/pages/3-dot-2-11-printing-formats) Write print statements using f-strings.

1. Download the PyPoll\_Challenge\_starter\_code.py file and rename it PyPoll\_Challenge.py.
2. Use the step-by-step instructions below to add code where indicated by the numbered comments in the starter code file.

**Step 1:**

* Initialize a county list, like the candidate\_options list, that will hold the names of the counties.
* Initialize a dictionary, like the candidate\_votes dictionary, that will hold the county as the key and the votes cast for each county as the values.

**Step 2:**

* Initialize an empty string, like winning\_candidate, that will hold the county name for the county with the largest turnout.
* Initialize a variable, like the winning\_count variable, that will hold the number of votes of the county that had the largest turnout.

**Step 3:**

* While reading the election results from each row inside the for loop, write a script that gets the county name from each row.

**Step 4a:**

* Write a decision statement with a logical operator to check if the county name acquired in Step 3 is in the county list you created in Step 1.

**Step 4b:**

* If the county is not in the list created in Step 1, add it to the list of county names like you did when adding a candidate to the candidate\_options list.

**Step 4c:**

* Write a script that initializes the county vote to zero, like you did when you began to track the vote counts for the candidates.

**Step 5:**

* Write a script that adds a vote to the county’s vote count as you are looping through all the rows, like you did for the candidate’s vote count.

**Step 6a:**

* Write a repetition statement to get the county from the county dictionary that was created in Step 1.

**Step 6b:**

* Initialize a variable to hold the county’s votes as they are retrieved from the county votes dictionary.

**Step 6c:**

* Write a script that calculates the county’s votes as a percentage of the total votes.

**Step 6d:**

* Write a print statement that prints the current county, its percentage of the total votes, and its total votes to the command line.

**Step 6e:** **This step will be completed in Deliverable 2.**

**Step 6f:**

* Write a decision statement that determines the county with the largest vote count and then adds that county and its vote count to the variables created in Step 2.

**Step 7:**

* Write a print statement that prints out the county with the largest turnout.

After you run your solution to Deliverable 1, confirm that the output to the command line matches the following image:

### Deliverable 1 Requirements

You will earn a perfect score for Deliverable 1 by completing all requirements below:

#### Candidate Results

* Total Votes in the election are printed to the terminal. **(5 pt)**
* Each candidate’s total votes and percentage of votes are printed to the terminal. **(5 pt)**
* The winner of the election, winning vote count, and winning percentage of votes are printed to the terminal. **(5 pt)**

#### County Results

* Each county and its total vote count are printed to the terminal. **(15 pt)**
* Each county and its percentage of the total votes are printed to the terminal. **(10 pt)**
* The county with the largest number of voters is printed to the terminal. **(10 pt)**

## Deliverable 2: Election Results Saved to a Text File (30 points)

### Deliverable 2 Instructions

Using your knowledge of writing data to a text file, write the winning candidate results and the county election results to the election\_results.txt file.

**REWIND**

For this deliverable, you’ve already done the following in this module:

* [**Lesson 3.2.2:**](https://courses.bootcampspot.com/courses/1225/pages/3-dot-2-2-execute-python-files) Run a Python file in command line or VS Code.
* [**Lesson 3.2.10:**](https://courses.bootcampspot.com/courses/1225/pages/3-dot-2-10-repetition-statements) Write data to a file.

Use the step-by-step instructions below to add code where indicated by the numbered comments in the starter code file.

**Step 6e:**

* Write a script that saves each county, the county’s total votes, and the county’s percentage of total votes to the election\_results.txt file.

**Step 8:**

* Write a script that saves the county with the largest turnout to the election\_results.txt file.

After you run your solution to Deliverable 2, confirm that your election\_results.txt file matches the following image:

### Deliverable 2 Requirements

You will earn a perfect score for Deliverable 2 by completing all requirements below:

#### Candidate Results

* Total Votes in the election are saved in the election\_results.txt file. **(2 pt)**
* Each candidate’s total votes and percentage of votes are saved in the election\_results.txt file. **(3 pt)**
* The winner of the election, winning vote count, and winning percentage of votes are saved in the election\_results.txt file. **(5 pt)**

#### County Results

* Each county and its total vote count are saved in the election\_results.txt file. **(10 pt)**
* Each county and its percentage of the total votes are saved in the election\_results.txt file. **(5 pt)**
* The county with the largest number of voters is saved in the election\_results.txt file. **(5 pt)**

## Deliverable 3: Written Analysis of the Election Audit (20 points)

### Deliverable 3 Instructions

Use your repository README to write your analysis of Deliverables 1 and 2. The analysis should contain the following:

1. **Overview of Election Audit:** Explain the purpose of this election audit analysis.
2. **Election-Audit Results:** Using a bulleted list, address the following election outcomes. Use images or examples of your code as support where necessary.
   * How many votes were cast in this congressional election?
   * Provide a breakdown of the number of votes and the percentage of total votes for each county in the precinct.
   * Which county had the largest number of votes?
   * Provide a breakdown of the number of votes and the percentage of the total votes each candidate received.
   * Which candidate won the election, what was their vote count, and what was their percentage of the total votes?
3. **Election-Audit Summary:** In a summary statement, provide a business proposal to the election commission on how this script can be used—with some modifications—for any election. Give at least two examples of how this script can be modified to be used for other elections.

### Deliverable 3 Requirements

#### Structure, Organization, and Formatting (6 points)

The written analysis has the following structure, organization, and formatting:

* There is a title, and there are multiple sections. **(2 pt)**
* Each section has a heading. **(2 pt)**
* Links to images are working, and code is formatted and displayed correctly. **(2 pt)**

#### Analysis (14 points)

The written analysis has the following:

* Overview of Election Audit
  + The purpose of this election analysis audit is well defined. **(3 pt)**
* Election Audit Results
  + There is a bulleted list where each election outcome is addressed. **(7 pt)**
* Election Audit Summary
  + There is a statement to the election commission that explores how this script can be used for any election, with two examples for modifying the script. **(4 pt)**

## **Submission**

Once you’re ready to submit, make sure to check your work against the rubric to ensure you are meeting the requirements for this Challenge one final time. It’s easy to overlook items when you’re in the zone!

As a reminder, the deliverables for this Challenge are as follows:

* Deliverable 1: The Election Results Printed to the Command Line
* Deliverable 2: The Election Results Saved to a Text File
* Deliverable 3: A written Analysis of the Election Audit (README.md)

Upload the following to your Election\_Analysis GitHub repository:

1. The PyPoll\_Challenge.py file
2. The analysis folder with the election\_results.txt file
3. The Resources folder with the election\_results.csv file

To submit your challenge assignment for grading in Bootcamp Spot, click Start Assignment, click the Website URL tab, then provide the URL of your Election\_Analysis GitHub repository, and then click Submit. Comments are disabled for graded submissions in BootCampSpot. If you have questions about your feedback, please notify your instructional staff or the Student Success Manager. If you would like to resubmit your work for an improved grade, you can use the **Re-Submit Assignment** button to upload new links. You may resubmit up to 3 times for a total of 4 submissions.

**IMPORTANT**

Once you receive feedback on your Challenge, make any suggested updates or adjustments to your work. Then, add this week’s Challenge to your professional portfolio.

**NOTE**

You are allowed to miss up to two Challenge assignments and still earn your certificate. If you complete all Challenge assignments, your lowest two grades will be dropped. If you wish to skip this assignment, click Next, and move on to the next Module.

## **Rubric**

**Module-3 Rubric**

| Module-3 Rubric | | |
| --- | --- | --- |
| **Criteria** | **Ratings** | **Pts** |
| This criterion is linked to a Learning OutcomeDeliverable 1: Election Results Printed to the Command Line | |  |  |  |  |  | | --- | --- | --- | --- | --- | | **50 to >46.0 pts**  **Demonstrating Proficiency**  ✓The Deliverable Fulfills "Emerging" Required Criteria. AND: ✓Each county and its total vote count are printed to the command line according to the solution. ✓Each county and its percent of the total votes are printed to the command line according to the solution. ✓The county with the largest number of voters is printed to the command line according to the solution. | **46 to >43.0 pts**  **Approaching Proficiency**  ✓The Deliverable Fulfills "Emerging" Required Criteria. AND: ✓Each county and its total vote count are printed to the command line according to the solution. ✓Each county and its percent of the total votes are printed to the command line with one or two errors. ✓The county with the largest number of voters is printed to the command line with one or two errors. | **43 to >39.0 pts**  **Developing Proficiency**  ✓The Deliverable Fulfills "Emerging" Required Criteria. AND: ✓Each county and its total vote count are printed to the command line, but there are one or two minor errors according to the solution. ✓Code is written to calculate the percent of the county votes with one error, but the results are not printed to the command line. ✓Code is written to determine the county with the largest number of votes with one error, but the results are not printed to the command line. | **39 to >0.0 pts**  **Emerging**  ✓All the following are printed to the command line: REQUIRED ✓Total Votes in the election. ✓Each candidate’s total votes and percent of votes. ✓The winner of the election, winning vote count, and winning percent of votes. AND: ✓Each county and its total vote count are printed to the command line, but not according to the solution. ✓Code is written to calculate the percent of the county votes with some errors, but the results are not printed to the command line. ✓Code is written to determine the county with the largest number of votes with some errors, but the results are not printed to the command line. | **0 pts**  **Incomplete** | | 50 pts |
| This criterion is linked to a Learning OutcomeDeliverable 2: Election Results Saved to a Text File | |  |  |  |  |  | | --- | --- | --- | --- | --- | | **30 to >25.0 pts**  **Demonstrating Proficiency**  ✓The Deliverable Fulfills "Emerging" Required Criteria. AND: ✓Each county and their total vote count are saved to the text according to the solution. ✓Each county and its percent of the total votes are saved to the text according to the solution. ✓The county with the largest number of voters is saved to the text according to the solution. | **25 to >22.0 pts**  **Approaching Proficiency**  ✓The Deliverable Fulfills "Emerging" Required Criteria. AND: ✓Each county and their total vote count are saved to the text file according to the solution with one minor error. ✓Code is written to save the percent of the county votes to the text file with one minor error. ✓Code is written to save the county with the largest number of votes to the text file with one minor error. | **22 to >20.0 pts**  **Developing Proficiency**  ✓The Deliverable Fulfills "Emerging" Required Criteria AND: ✓Each county and their total vote count are saved to the text according to the solution but with one or two minor errors. ✓Code is written to save the percent of the county votes to a text file with one or two minor errors. ✓Code is written to save the county with the largest number of votes to the text file with one or two minor errors. | **20 to >0.0 pts**  **Emerging**  ✓All of the following saved to the text file: REQUIRED ✓Total Votes in the election. ✓Each candidate’s total votes and percent of votes. ✓The winner of the election, winning vote count, and winning percent of votes. AND: ✓Each county and their total vote count are saved with some errors, and not according to the solution. ✓Code is written to save the percent of the county votes to the text file but the code is not working. ✓Code is written to save the county with the largest number of votes to the text file but the code is not working. | **0 pts**  **Incomplete** | | 30 pts |
| This criterion is linked to a Learning OutcomeDeliverable 3: Structure, Organization, and Formatting | |  |  |  |  |  | | --- | --- | --- | --- | --- | | **6 to >5.0 pts**  **Demonstrating Proficiency**  ✓The written analysis has ALL of the following: ✓There is a title, and there are multiple sections. ✓Each section has a heading. ✓There are images and references to code, and they are formatted and displayed correctly. | **5 to >3.0 pts**  **Approaching Proficiency**  ✓The written analysis has ALL of the following: ✓There is a title, and there are multiple sections. ✓Each section has a heading. ✓There are images and references to code, and they are formatted and displayed correctly with one or two minor errors. | **3 to >2.0 pts**  **Developing Proficiency**  ✓The written analysis has ALL of the following: ✓There is a title, and there are multiple sections. AND ONE of the following: ✓Each section may have a heading. ✓There are images and references to code, and they are formatted and displayed correctly with one or two minor errors. | **2 to >0.0 pts**  **Emerging**  ✓The written analysis has ALL of the following: ✓There is a title. ✓There are no headings for each section, but there are three sections. | **0 pts**  **Incomplete** | | 6 pts |
| This criterion is linked to a Learning OutcomeDeliverable 3: Analysis | |  |  |  |  |  | | --- | --- | --- | --- | --- | | **14 to >13.0 pts**  **Demonstrating Proficiency**  ✓The purpose is well defined. ✓ALL FIVE election outcomes are addressed. ✓There is a statement to the election commission on how this script can be used for any election with two examples given. | **13 to >11.0 pts**  **Approaching Proficiency**  ✓The purpose is well defined. ✓FOUR of the FIVE election outcomes are addressed. ✓There is a statement to the election commission on how this script can be used for any election with two examples given | **11 to >9.0 pts**  **Developing Proficiency**  ✓The purpose is well defined. ✓THREE to FOUR of the FIVE election outcomes are addressed. ✓There is a statement to the election commission on how this script can be used for any election with two examples given. | **9 to >0.0 pts**  **Emerging**  ✓The purpose is well defined. ✓TWO of the FIVE election outcomes are addressed. ✓There is a statement to the election commission on how this script can be used for any election with only one to two examples given. | **0 pts**  **Incomplete** | | 14 pts |
| Total Points: 100 | | |

[Previous](https://courses.bootcampspot.com/courses/1225/modules/items/497259)[Next](https://courses.bootcampspot.com/courses/1225/modules/items/497273)

© 2020 - 2022 Trilogy Education Services, a 2U, Inc. brand. All Rights Reserved.